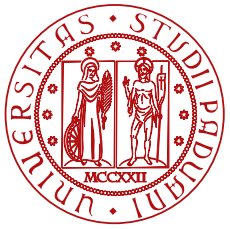


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Control System Engineering

Implementation of adaptive nonlinear model predictive control on a PX4-enabled quad-rotor platform

Master Candidate

Marco Concetto Bonazza

Student ID 2027593

Supervisor

Prof. Angelo Cenedese

University of Padova

Co-supervisor

Dott. Beniamino Pozzan

University of Padova

Academic Year - 2022/2023

Graduation Date - 20/04/2023

Abstract

This thesis aims at developing an adaptive and nonlinear model predictive control Simulink scheme and interfacing it with the popular PX4 drone system. PX4-Autopilot is one of the most used drone Real Time Operating System (RTOS) in the context of research, it has many safety and sensor management features, it is open source, and has an extensive and active community of developers making it an excellent platform for Unmanned Aerial Vehicles (UAVs) control development. The advantages of interfacing it with MATLAB®/Simulink® running on a companion computer are mainly twofold. The first is simplicity: the Simulink block scheme language is easy to use for complex control schemes, also supported by a great collection of libraries and by the baked-in management of PX4 of sensor data that can directly be used as feedback for the controls without additional estimators. The second is the possibility of moving the computational complexity away from the onboard embedded platform to a much more powerful ground station PC. Nonlinear Model Predictive Control (NMPC) is an excellent example as it makes use of both, there are many implementations available that require only some setup and the model of the plant, it gives great control performance but is computationally expensive and therefore not always usable directly on low-end embedded hardware without some optimizations, which would require a competent and experienced user. Since model predictive control is susceptible to modeling errors that are especially common when dealing with low-cost drone platforms it is paired with a lightweight adaptive scheme that complements the control action to make up for modeling mismatches. The whole infrastructure is then validated through Software In the Loop (SITL) simulations across a variety of tasks and conditions, confirming that the interface between MATLAB/Simulink works, the NMPC scheme is usable in real-time with good trajectory tracking performance and that adaptive control provides a much greater degree of robustness to the system.

Keywords: UAV, Quadrotor, Nonlinear Model Predictive Control, L1 Adaptive Control, Ground Effect, PX4

Contents

List of Figures

List of Tables

List of Acronyms

w.r.t. With Respect To

s.t. Such That

i.e. Id Est, that is

MEMS Micro Electro Mechanical Systems

VTOL Vertical TakeOff Landing

DOFs Degrees Of Freedom

CoM Center of Mass

FLU Front-Left-Up

ENU East-North-Up

FRD Front-Right-Down

NED North-East-Down

CCW CounterClockWise

CW ClockWise

LQR Linear Quadratic Regulator

MIMO Multiple-Input Multiple-Output

RHC Receding Horizon Control

MPC Model Predictive Control

NMPC Nonlinear Model Predictive Control

UAVs Unmanned Aerial Vehicles

ARE Algebraic Riccati Equation

MRAC Model Reference Adaptive Control

APIs Application Programming Interfaces

SDKs Software Development Kits

VTOL Vertical TakeOff and Landing

IMU Inertial Measurement Unit

RC Radio Command

EKF Extended Kalman Filter

ESCs Electronic Speed Controllers

QGC QGround Control

SITL Software In the Loop

HITL Hardware In the Loop

SQP Sequential Quadratic Programming

API Application Programming Interface

RMSE Root Mean Square Error

RTOS Real Time Operating System

1 Introduction

In recent years thanks to the development of microprocessors and Micro Electro Mechanical Systems (MEMS) the UAVs field has been a very fertile area of research. In particular, quad-copters are widely studied and used in both consumer and industrial applications that range from terrain surveying for agriculture, rescue missions, cinematography and so many more. The main advantage of these drones are simple mechanical structure, Vertical TakeOff and Landing (VTOL) and hovering capabilities [1]. There are also some major drawbacks: they suffer from under actuation and heavy nonlinearities, so much so that classical control solutions usually are not able to fully utilize their capabilities since they need linearizations around operating conditions and small-angle-like assumptions for control design [2], [3].

A particularly interesting research topic on these vehicles that tries to avoid those problems is nonlinear model predictive control. Even though computationally expensive, it is a very popular control algorithm since only the model of the system is necessary, eliminating the need to explicitly design a control law, shifting the effort to accurate model construction [4]. It provides explicit actuation and state constraint handling and a direct physical interpretation of the tuning parameters [5]. Its main drawback is that performance is heavily dependent on the quality of the model, every approximation could influence the nonlinear dynamics such that the forecasted states are not relevant anymore, throwing off the optimality of the control input.

Due to the low-cost nature of most qua-rotor platforms, the manufacturing tolerances are not always strict, leading to inaccurate models that would be too costly or time-consuming to correct. Thus the need for robust control algorithms [6], [7]. In this thesis to mitigate the modeling problem, NMPC is implemented alongside an adaptive scheme that provides a purely reactive and systematic approach for compensating modeling mismatches in real-time called $\mathcal{L}_1$. It is an extension of the state-predictor-based MRAC (Model Reference Adaptive Control) algorithm that eliminates the sensibility to high-frequency excitations by adding

a low pass filter to the output. This creates an explicit and easily tunable tradeoff between robustness and performance [8], [9].

Although flight controllers are becoming more and more advanced they still present a not negligible obstacle when trying to implement more complex control algorithms and thus the objective of this work is to design, implement and test a state-of-the-art NMPC algorithm with an additive $\mathcal{L}_1$ component that runs in MATLAB/Simulink on a ground-based PC that is able to interface, send commands and receive vehicle odometry from a PX4-enabled quadrotor. PX4 is one of the most popular open-source drone RTOS used in research context, it provides a plethora of safety features, sensor and battery management, great customization and thanks to its optimized code-base it can run smoothly on embedded hardware.

1.1 Thesis structure

This work is organized into the following chapters:

- In Chapter 2 the preliminaries on rotations in 3d space are given, both in terms of rotation matrices and quaternions. Then the mathematical model for the quad-rotor is explained and derived using the Newton-Euler algorithm.
- In Chapter 3 the concept of optimal control is presented first since it is the basis for the development of receding horizon and model predictive control. The latter is then briefly presented in terms of core ideas. Finally, the main ingredients of nonlinear model predictive control in the case of a quad-rotor platform are presented and discussed.
- In Chapter 4 model reference adaptive control is presented and its main features and drawbacks are discussed together with the reason behind its combination with NMPC. Then the $\mathcal{L}_1$ piece-wise constant adaptation law is derived in the context of quad-rotors.
- In Chapter 5 there is an overview of the PX4 software features, structure and means of connection to MATLAB/Simulink. Then there is a high-level description of the Stateflow machine that manages communication with PX4 and the execution of the Simulink scheme. At the end of the chapter, there is an important distinction between the different reference frames used in this work and the standard ones used in PX4.
- In Chapter 6 the simulation environment is briefly presented and then augmented with a ground effect model to simulate the increased thrust phenomenon that happens when a pocket of air forms under a flying vehicle. Finally, some SITL simulations are carried out using different tasks and

modeling error conditions to validate the effectiveness of the complete architecture.

- In chapter 7 the objective and results of this work are summarized, then suggestions for future work are provided.

2 Agent Modelling

This chapter provides a preliminary explanation of how and with which conventions the pose of a rigid body is described in 3D space, then the generic dynamic model for a quadrotor in x configuration is presented.

2.1 Pose representation

In 3D space, a rigid object has 6 Degrees Of Freedom (DOFs) that describe completely its pose, three for its position and three for its orientation. To describe the condition of said system it is necessary to introduce at least two reference frames, $\mathcal{F}_W$ with origin O_W and $\mathcal{F}_B$ with origin O_B, which are respectively fixed to the ground (world-frame) and fixed to the Center of Mass (CoM) of the object (body-frame) so that the latter can be defined in relation to the former, see Fig 2.1.

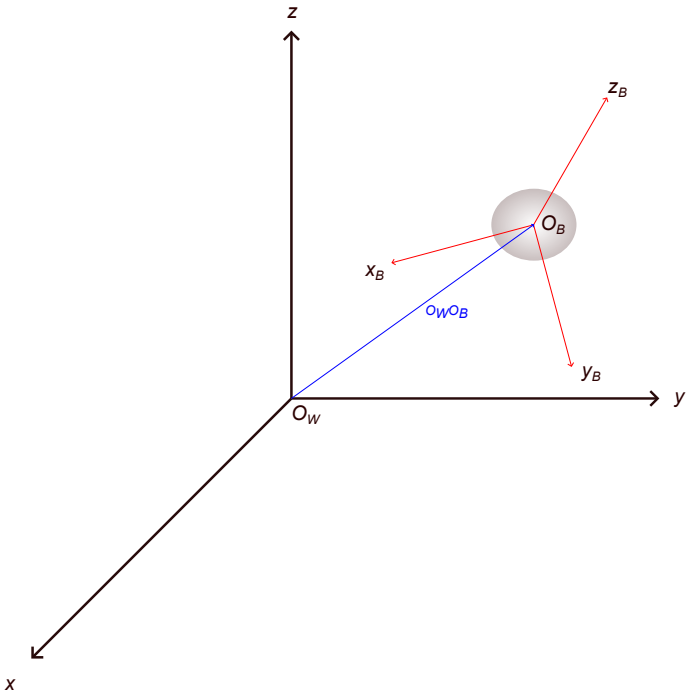


Figure 2.1: World and Body fixed frames of reference.

The position of the origin of the body-frame w.r.t. the origin of the world frame, expressed in world frame, is given by the vector

$$\mathbf{p} = O_W O_B = \begin{bmatrix} O_B^x \\ O_B^y \\ O_B^z \end{bmatrix} \in \mathbb{R}^3 \tag{2.1}$$

Relative orientation description is much more ambiguous because it can be expressed with multiple formalisms and conventions, the main ones that are used throughout this work are presented in the subsections below.

2.1.1 Rotation matrices

3D Rotation matrices, as the name implies are matrices that express the relative orientation between the fixed reference frame $\mathcal{F}_W$ and the body-fixed one $\mathcal{F}_B$. They belong to the Special Orthogonal group $\mathbb{SO}(3)$, meaning that some important properties apply, namely for a matrix $\mathbf{R} \in \mathbb{SO}(3)$ it holds

$$det(\mathbf{R}) = 1 \tag{2.2a}$$

$$\mathbf{R}\mathbf{R}^\top = I_{3\times 3} \tag{2.2b}$$

$$\mathbf{R}^{-1} = \mathbf{R}^\top \tag{2.2c}$$

These properties have particular physical significance, in particular (2.2a) and (2.2b) indicate that distances, lines, angles and areas are preserved and not mirrored.

The easiest conceptual way of constructing the matrix that expresses the rotation from the body-frame to the world-frame is to express the coordinates of the versors

of $\mathcal{F}_B$ w.r.t. $\mathcal{F}_W$, more formally

$$\begin{aligned}\mathbf{R}_{WB} &= \begin{bmatrix} \hat{\mathbf{i}}_B & \hat{\mathbf{j}}_B & \hat{\mathbf{k}}_B \end{bmatrix} = \begin{bmatrix} \hat{i}_{Bx} & \hat{j}_{Bx} & \hat{k}_{Bx} \\ \hat{i}_{By} & \hat{j}_{By} & \hat{k}_{By} \\ \hat{i}_{Bz} & \hat{j}_{Bz} & \hat{k}_{Bz} \end{bmatrix} \\ &= \begin{bmatrix} < \hat{\mathbf{i}}_B, \hat{\mathbf{i}} > & \hat{\mathbf{j}}_B, \hat{\mathbf{i}} > & < \hat{\mathbf{k}}_B, \hat{\mathbf{i}} > \\ < \hat{\mathbf{i}}_B, \hat{\mathbf{j}} > & \hat{\mathbf{j}}_B, \hat{\mathbf{j}} > & < \hat{\mathbf{k}}_B, \hat{\mathbf{j}} > \\ < \hat{\mathbf{i}}_B, \hat{\mathbf{k}} > & \hat{\mathbf{j}}_B, \hat{\mathbf{k}} > & < \hat{\mathbf{k}}_B, \hat{\mathbf{k}} > \end{bmatrix}\end{aligned} \tag{2.3}$$

Where, for example, $\hat{}_{Bx}, \hat{}_{By}, \hat{}_{Bz}$ are the components of the body axes versors w.r.t. the fixed frame, $\hat{\mathbf{i}}, \hat{\mathbf{j}}, \hat{\mathbf{k}}$ are the fixed frame versors and $< \cdot, \cdot >$ is the scalar product between vectors. A simplified example can be seen in Fig.2.2.

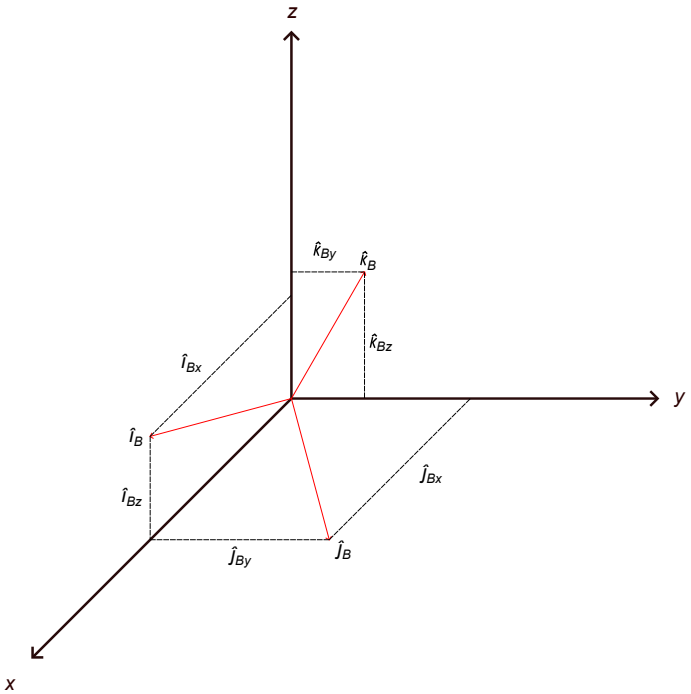


Figure 2.2: Example of projection of Body rotated frame axes on World fixed frame axes, where each body axis belongs to the plane formed by the other two world axes to simplify the drawing.

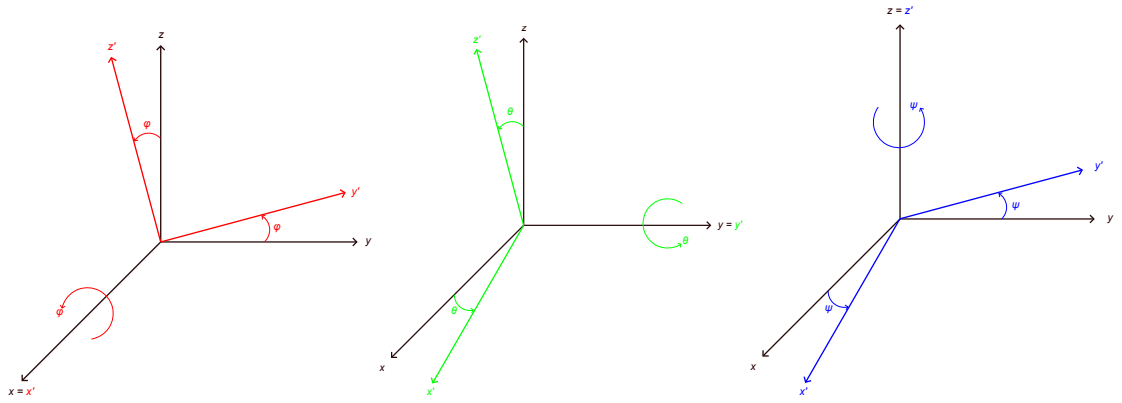
Finally, a vector $\mathbf{P}_B$ expressed in body-frame can be rotated through $\mathbf{R}_{WB}$ to get the following relations with the same vector expressed in the world-frame $\mathbf{P}_W$

$$\mathbf{P}_W = \mathbf{R}_{WB}\mathbf{P}_B \tag{2.4a}$$

$$\mathbf{P}_B = \mathbf{R}_{BW}\mathbf{P}_W = \mathbf{R}_{WB}^{-1}\mathbf{P}_W = \mathbf{R}_{WB}^{\top}\mathbf{P}_W \tag{2.4b}$$

Elementary rotations and Euler angles

Any rotation matrix needs nine parameters to be fully defined but due to the orthogonality constraints, only three of those are really needed. Euler's rotation theorem states that a generic rotation matrix can be obtained by composing a suitable sequence of three elementary rotations (Fig. 2.3) while guaranteeing that two successive rotations are not made about parallel axes. Elementary rotations are thus essential to describe an arbitrary rotation easily.



(a) (b) (c)

Figure 2.3: (a) Rotation of ϕ around x, (b) Rotation of θ around y, (c) Rotation of ψ around z.

They can be found by keeping one of the three axes fixed and rotating the other two around it

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \tag{2.5a}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{2.5b}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5c}$$

The second part of the theorem limits the combinations of the three rotations to 12 possible conventions, of which the Euler ZYX, or equivalently but with a different name convention roll-pitch-yaw XYZ is used throughout this work and is presented below. The difference in naming convention stems from the fact that the final rotation matrix is obtained by pre-multiplication of the elementary ones so the order of multiplication is Z then Y then X, while the order of rotation is X then Y then Z. To simplify notation $cos(\cdot) \equiv c.$ and $sin(\cdot) \equiv s.$

$$\begin{aligned} \mathbf{R}_{WB}(\phi, \theta, \psi) &= \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi) \\ &= \begin{bmatrix} c_\psi c_\theta & -s_\psi c_\phi + c_\psi s_\theta s_\phi & s_\psi s_\phi + c_\psi s_\theta c_\phi \\ s_\psi c_\theta & c_\psi c_\phi + s_\psi s_\theta s_\phi & -c_\psi s_\phi + s_\psi s_\theta c_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \end{aligned} \tag{2.6}$$

If not in a condition called gymbal-lock ($\theta \neq \pm\frac{\pi}{2}$) the angles can be retrieved from the final rotation matrix using

$$\phi = atan2(r_{32}, r_{33}) \tag{2.7a}$$

$$\theta = asin(-r_{31}) \tag{2.7b}$$

$$\psi = atan2(r_{21}, r_{11}) \tag{2.7c}$$

Where r_{ab} represents the element of the rotation matrix in position (a, b). It is important to note that if in gymbal lock only $\phi \pm \psi$ can be retrieved.

This convention is often used in relation to air or seaborne vehicles because it allows an easy understanding of the changes in orientation that each angle causes (Fig. 2.4); specifically the rotation about the x-axis (roll) indicates how inclined the sides of the vehicle are w.r.t. the horizontal plane, the one about the y axis (pitch) indicates how much the front of the vehicle is inclined up or down w.r.t. the horizontal plane and finally the rotation about the z-axis (yaw) indicates the direction in which the front of the vehicle is pointed in the xy plane.

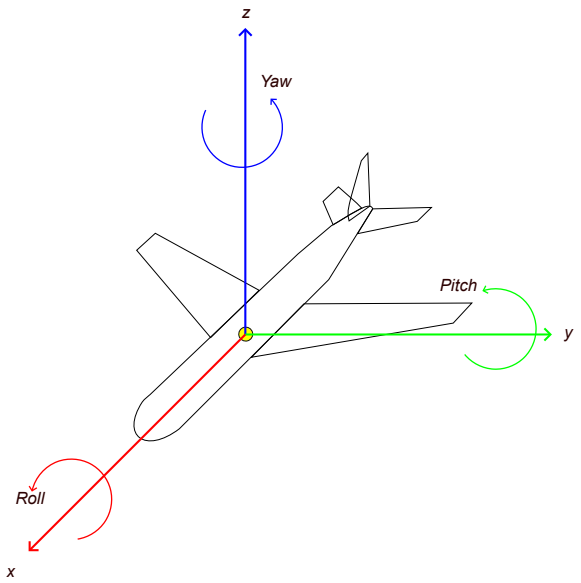


Figure 2.4: Physical meaning associated with Roll-Pitch-Yaw convention.

2.1.2 Unit quaternions

Quaternions are hyper-complex numbers proposed by William Rowan Hamilton in 1843 as a higher dimensional extension of complex numbers.

Today quaternions with unitary norm, or unit-quaternions, see great use as an alternative representation for $\mathbb{SO}(3)$ rotations since, unlike Euler angles, they do not suffer from gymbal lock, provide a more computationally efficient way of computing rotations (fewer operations are needed to rotate a vector w.r.t. rotation matrices) and if during calculations rounding errors pile up they can be normalized and still represent rotations, while rotation matrices could lose orthogonality.

Unit quaternions are composed by a real part η and a complex vector part $\boldsymbol{\epsilon}$ so that

$$\mathbf{q} = \begin{bmatrix} \eta \\ \epsilon_i \\ \epsilon_j \\ \epsilon_k \end{bmatrix} = \begin{bmatrix} \eta \\ \boldsymbol{\epsilon} \end{bmatrix} \tag{2.8a}$$

$$\|\mathbf{q}\|^2 = \eta^2 + \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon} = 1 \tag{2.8b}$$

Where (i, j, k) is a coordinate frame that follows Hamilton's rules

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2.9a}$$

$$ij = k, jk = i, ki = j \tag{2.9b}$$

$$ji = -k, kj = -i, ik = -j \tag{2.9c}$$

Unit quaternions have a physical interpretation if considering a rotation of an angle α around an axis represented by the unit vector $\mathbf{e} \in \mathbb{R}(3)$ in the following way

$$\mathbf{q} = \begin{bmatrix} \eta \\ \boldsymbol{\epsilon} \end{bmatrix} = \begin{bmatrix} cos(\frac{\alpha}{2}) \\ \mathbf{e} sin(\frac{\alpha}{2}) \end{bmatrix} \tag{2.10}$$

It is evident from (2.10) that the same rotation can be represented also by $-\mathbf{q}$: this property is called double coverage.

Quaternion conjugate is defined as

$$\overline{\mathbf{q}} = \begin{bmatrix} \eta \\ -\boldsymbol{\epsilon} \end{bmatrix} \tag{2.11}$$

This allows to introduce the quaternion inverse that, in the case of unit quaternions, corresponds to a rotation around the same axis with opposite angle and thus can be defined as

$$\mathbf{q}^{-1} = \overline{\mathbf{q}} \tag{2.12}$$

Before defining the relation between a vector and its rotated representations some preliminary definitions must be given on quaternion products:

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = \eta_1\eta_2 + \boldsymbol{\epsilon}_1 \cdot \boldsymbol{\epsilon}_2 = \eta_1\eta_2 + \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_2 \in \mathbb{R} \tag{2.13a}$$

$$\mathbf{q}_1 \times \mathbf{q}_2 = \eta_1\boldsymbol{\epsilon}_2 + \eta_2\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1 \times \boldsymbol{\epsilon}_2 = \eta_1\boldsymbol{\epsilon}_2 + \eta_2\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1 \times \boldsymbol{\epsilon}_2 \in \mathbb{R}^3 \tag{2.13b}$$

Where (2.13a) and (2.13b) represent quaternion inner and outer products respectively.

Rotation combination is defined by quaternion composition or Hamilton product as

$$\begin{aligned} \mathbf{q}_{tot} = \mathbf{q}_1 \circ \mathbf{q}_2 &= \begin{bmatrix} \eta_1 \\ \boldsymbol{\epsilon}_1 \end{bmatrix} \circ \begin{bmatrix} \eta_2 \\ \boldsymbol{\epsilon}_2 \end{bmatrix} \\ &= \begin{bmatrix} \eta_1\eta_2 - \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_2 \\ \eta_1\boldsymbol{\epsilon}_2 + \eta_2\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1 \times \boldsymbol{\epsilon}_2 \end{bmatrix} = \begin{bmatrix} \eta_2\eta_1 - \boldsymbol{\epsilon}_2^\top \boldsymbol{\epsilon}_1 \\ \eta_2\boldsymbol{\epsilon}_1 + \eta_1\boldsymbol{\epsilon}_2 - \boldsymbol{\epsilon}_2 \times \boldsymbol{\epsilon}_1 \end{bmatrix} \\ &= \begin{bmatrix} \eta_1 & -\boldsymbol{\epsilon}_1^\top \\ \boldsymbol{\epsilon}_1 & \eta_1 I_{3\times 3} + [\boldsymbol{\epsilon}_1]_\times \end{bmatrix} \begin{bmatrix} \eta_2 \\ \boldsymbol{\epsilon}_2 \end{bmatrix} = \begin{bmatrix} \eta_2 & -\boldsymbol{\epsilon}_2^\top \\ \boldsymbol{\epsilon}_2 & \eta_2 I_{3\times 3} - [\boldsymbol{\epsilon}_2]_\times \end{bmatrix} \begin{bmatrix} \eta_1 \\ \boldsymbol{\epsilon}_1 \end{bmatrix} \\ &= \mathbf{M}(\mathbf{q}_1)\mathbf{q}_2 = \mathbf{N}(\mathbf{q}_2)\mathbf{q}_1 \end{aligned} \tag{2.14}$$

Where $[\cdot]_\times$ is the skew-symmetric operator defined in (A.4).

Finally, to rotate a vector $\mathbf{P}_B$ expressed in body-frame it has to be first written as a purely imaginary quaternion $\widehat{\mathbf{P}_B} = \begin{bmatrix} 0 & \mathbf{P}_B^\top \end{bmatrix}^\top$, then to get the same vector expressed in the world-frame $\widehat{\mathbf{P}_W} = \begin{bmatrix} 0 & \mathbf{P}_W^\top \end{bmatrix}^\top$ the following composition has to be computed

$$\begin{aligned} \widehat{\mathbf{P}_W} &= \mathbf{q}_{WB} \circ \widehat{\mathbf{P}_B} \circ \mathbf{q}_{BW} = \mathbf{q}_{WB} \circ \widehat{\mathbf{P}_B} \circ \overline{\mathbf{q}}_{WB} \\ &= \mathbf{q}_{WB} \circ \begin{bmatrix} 0 \\ \mathbf{P}_B \end{bmatrix} \circ \overline{\mathbf{q}}_{WB} \\ &= \mathbf{M}(\mathbf{q}_{WB})\mathbf{N}(\overline{\mathbf{q}}_{WB}) \begin{bmatrix} 0 \\ \mathbf{P}_B \end{bmatrix} \end{aligned} \tag{2.15a}$$

$$\begin{aligned} \widehat{\mathbf{P}_B} &= \mathbf{q}_{BW} \circ \widehat{\mathbf{P}_W} \circ \mathbf{q}_{WB} = \overline{\mathbf{q}}_{WB} \circ \widehat{\mathbf{P}_W} \circ \mathbf{q}_{WB} \\ &= \mathbf{M}(\overline{\mathbf{q}}_{WB})\mathbf{N}(\mathbf{q}_{WB}) \begin{bmatrix} 0 \\ \mathbf{P}_W \end{bmatrix} \end{aligned} \tag{2.15b}$$

Where $\mathbf{q}_{WB}$ and $\mathbf{q}_{BW}$ are respectively the quaternions that encode the rotation from body-frame $\mathcal{F}_B$ to $\mathcal{F}_W$ and vice-versa.

Relationship with rotation matrices and Euler angles

Since unit quaternions and rotation matrices both represent rotations it is possible to go from the first to the second using Rodrigues formula, trigonometric half-angle formulas and recalling (2.10)

$$\begin{aligned}\mathbf{R}(\mathbf{q}) &= I_{3\times3} + 2\eta[\boldsymbol{\epsilon}]_\times + 2[\boldsymbol{\epsilon}]_\times^2 \\ &= \begin{bmatrix} \eta^2 + \epsilon_i^2 - \epsilon_j^2 - \epsilon_k^2 & 2\epsilon_i\epsilon_j - 2\eta\epsilon_k & 2\epsilon_i\epsilon_k + 2\eta\epsilon_j \\ 2\epsilon_i\epsilon_j + 2\eta\epsilon_k & \eta^2 - \epsilon_i^2 + \epsilon_j^2 - \epsilon_k^2 & 2\epsilon_j\epsilon_k - 2\eta\epsilon_i \\ 2\epsilon_i\epsilon_k - 2\eta\epsilon_j & 2\epsilon_j\epsilon_k + 2\eta\epsilon_i & \eta^2 - \epsilon_i^2 - \epsilon_j^2 + \epsilon_k^2 \end{bmatrix}\end{aligned} \tag{2.16}$$

Finally Euler angles can be retrieved by using (2.7) and thus

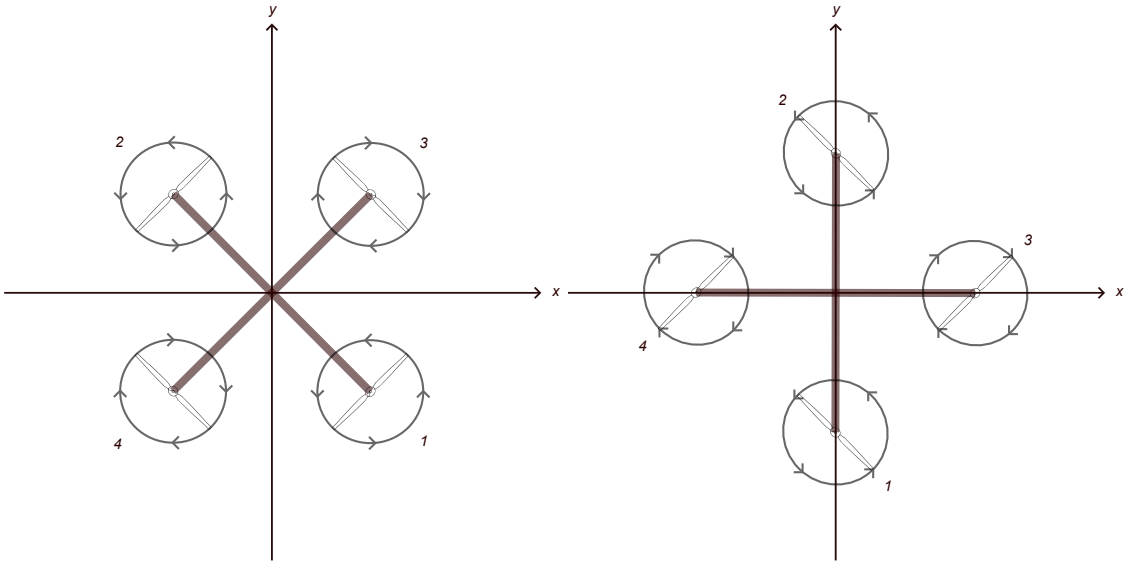
$$\phi = atan2(2\epsilon_j\epsilon_k + 2\eta\epsilon_i, \eta^2 - \epsilon_i^2 - \epsilon_j^2 + \epsilon_k^2) \tag{2.17a}$$

$$\theta = asin(-2\epsilon_i\epsilon_k + 2\eta\epsilon_j) \tag{2.17b}$$

$$\psi = atan2(2\epsilon_i\epsilon_j + 2\eta\epsilon_k, \eta^2 + \epsilon_i^2 - \epsilon_j^2 - \epsilon_k^2) \tag{2.17c}$$

2.2 Quadrotor mathematical model

In this section, the kinematic and dynamic models of a quadrotor that will be used to develop control schemes in the following chapters are given.



(a) (b)

Figure 2.5: Quadrotor X (a) and + (b) configurations.

The standard quadrotor is composed of a rigid frame with four propellers spinning about their own axis. The main configurations are represented in Fig.2.5: the first is called X configuration because the quadrotor's arms form 45° angles with the body frame and the second is called + configuration because the arms are aligned with the body axes. In this work, only the one in Fig.2.5a is considered because in contrast with the plus configuration, for the same desired motion, the cross-style provides higher momentum, which can increase the maneuverability performance as each move requires all four blades to vary their rotation speed [10].

To describe its model let's introduce (as done in Sec.2.1, Fig.2.1) the body frame $\mathcal{F}_B$, whose origin O_B coincides with the CoM of the platform, and the inertial world frame $\mathcal{F}_W$. It will be discussed in more detail in a following chapter in subsec.5.5.4 but it is worth mentioning that the model derived here uses the Front-Left-Up (FLU) or East-North-Up (ENU) conventions for its reference frames, whereas the PX4 environment uses the North-East-Down (NED) or Front-Right-Down (FRD) ones.

The pose of $\mathcal{F}_B$ w.r.t. $\mathcal{F}_W$ is fully defined by the pair $(\mathbf{p}, \mathbf{R}_B) \in \mathbb{R}^3 \times \mathbb{SO}(3)$ where the vector $\mathbf{p} \in \mathbb{R}^3$ and the rotation matrix $\mathbf{R}_{WB} \in \mathbb{SO}(3)$ represent respectively the position of O_B and the orientation of $\mathcal{F}_B$ w.r.t. $\mathcal{F}_W$.

By introducing the linear velocity $\mathbf{v} \in \mathbb{R}^3$ of O_B w.r.t. $\mathcal{F}_W$ and expressed in $\mathcal{F}_W$ and the angular velocity $\boldsymbol{\omega}_{WB} \in \mathbb{R}^3$ of $\mathcal{F}_B$ w.r.t. $\mathcal{F}_W$ but expressed in $\mathcal{F}_B$, also the twist of the quad-rotor platform can be fully defined by the pair $(\mathbf{v}, \boldsymbol{\omega}_B)$.

Thus the kinematic model of the quad-rotor is governed by the relations

$$\dot{\mathbf{p}} = \mathbf{v} \tag{2.18a}$$

$$\dot{\mathbf{R}}_{WB} = \mathbf{R}_{WB}[\boldsymbol{\omega}_B]_\times \tag{2.18b}$$

where the second relation is derived in sec.A.1, equation (A.11).

The dynamic model can be derived using two main formalisms: Euler-Lagrange, which is energy-based and more compact but less intuitive, and the Newton-Euler which is based directly on the forces and moments acting on the system. In this thesis, only the latter is considered for the model derivation since it is easier to understand and more commonly found in the literature. A visual representation of all forces and moments acting on the system is reported in Fig.2.6 and explained one by one below.

The main forces (apart from gravity) and torques acting on the quadrotor are generated by each of its propellers, spun by their respective motors. The i-th propeller, with $i = 1, 2, 3, 4$, rotates around its own spinning axis passing through its center O_{P_i} and parallel to the z body-frame axis, with a controllable angular rate $\omega_i \in \mathbb{R}$. Depending on the direction in which the propeller is spinning the angular velocity vector changes direction, it is directed along the positive direction of the spinning axis if the rotation is CounterClockWise (CCW), in the negative if the rotation is ClockWise (CW). The thrust generated by the spinning propellers is proportional to the squared angular velocity of the motor so the control input can be defined as $u_i = \omega_i^2 \in \mathbb{R}$.

Thus each propeller with its rotation creates several effects:

A thrust force, expressed in $\mathcal{F}_B$, equal to

$$\mathbf{f}_i = c_{f_i} u_i \widehat{\mathbf{k}}_{B_i} \in \mathbb{R}^3 \tag{2.19}$$

Where $c_{f_i} \in \mathbb{R}^+$ is a constant aerodynamic parameter that expresses the relationship between the squared spinning rate and upward thrust of the i-th propeller.

$\widehat{\mathbf{k}}_{B_i}$ is a versor parallel to the z body-frame axis that passes through O_{P_i}.

A force moment, expressed in $\mathcal{F}_B$, generated by the thrust force on the body equal to

$$\boldsymbol{\tau}_i^t = \mathbf{p}_i \times \mathbf{f}_i = \mathbf{p}_i \times c_{f_i} u_i \widehat{\mathbf{k}}_{B_i} \in \mathbb{R}^3 \tag{2.20}$$

where $\mathbf{p}_i \in \mathbb{R}^3$ is the vector that encodes the position of O_{P_i} w.r.t. the origin of the body frame, expressed in body frame.

A drag torque, expressed in $\mathcal{F}_B$, generated by air friction, which is opposite to the angular velocity of each propeller and is equal to

$$\boldsymbol{\tau}_i^d = c_{\tau_i} u_i \widehat{\mathbf{k}}_{B_i} \in \mathbb{R}^3 \tag{2.21}$$

where $c_{\tau_i} \in \mathbb{R}$ is a constant aerodynamic parameter that expresses the relationship between squared spinning rate and rotational air drag; it is positive if the rotor is spinning clockwise, negative otherwise.

Finally by using (2.19), (2.20), (2.21) the expression in body-frame of the total force $\mathbf{f}_c \in \mathbb{R}^3$ and total torque $\boldsymbol{\tau}_c \in \mathbb{R}^3$ that act on O_B, which is the CoM of the

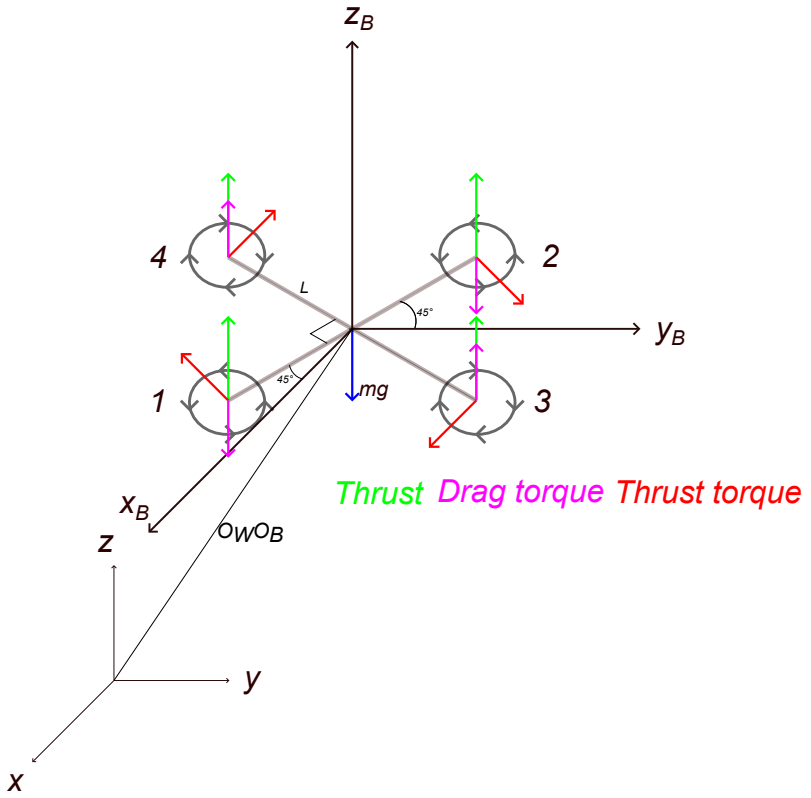


Figure 2.6: Forces and moments acting on the quadrotor.

quadrotor, can be found as

$$\mathbf{f}_c = \sum_{i=1}^{4} \mathbf{f}_i = \sum_{i=1}^{4} c_{f_i} u_i \widehat{\mathbf{k}}_{B_i} \tag{2.22a}$$

$$\boldsymbol{\tau}_c = \sum_{i=1}^{4} (\boldsymbol{\tau}_i^t + \boldsymbol{\tau}_i^d) = \sum_{i=1}^{4} (c_{f_i} \mathbf{p}_i \times \widehat{\mathbf{k}}_{B_i} + c_{\tau_i} \widehat{\mathbf{k}}_{B_i}) u_i \tag{2.22b}$$

These equations can be shortened by introducing the matrices $\mathbf{F} \in \mathbb{R}^{3\times4}$ and $\mathbf{M} \in \mathbb{R}^{3\times4}$, which are called respectively control force and control moment input matrices, and the input vector $\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^\top \in \mathbb{R}^4$

$$\mathbf{f}_c = \mathbf{F}\mathbf{u} \tag{2.23a}$$

$$\boldsymbol{\tau}_c = \mathbf{M}\mathbf{u} \tag{2.23b}$$

where, assuming for simplicity's sake that all arms of the quadrotor have the same length L and they form angles of 45° with each other, the position of each rotor w.r.t. $\mathcal{F}_B$ can be expressed as $\begin{bmatrix} \pm L cos(45°) & \pm L sin(45°) & 0 \end{bmatrix}^\top = \begin{bmatrix} \pm L_x & \pm L_y & 0 \end{bmatrix}^\top$ so that **F** and **M** assume the following expressions

$$\mathbf{F} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ c_{f_1} & c_{f_2} & c_{f_3} & c_{f_4} \end{bmatrix} \tag{2.24a}$$

$$\mathbf{M} = \begin{bmatrix} -c_{f_1} L_y & c_{f_2} L_y & c_{f_3} L_y & -c_{f_4} L_y \\ -c_{f_1} L_x & c_{f_2} L_x & -c_{f_3} L_x & c_{f_4} L_x \\ -|c_{\tau_1}| & -|c_{\tau_2}| & |c_{\tau_3}| & |c_{\tau_4}| \end{bmatrix} \tag{2.24b}$$

Again, for simplicity, all second-order effects such as the gyroscopic and inertial effects and also some aerodynamic phenomena like propeller blade flapping are neglected. The dynamics of the quadrotor, expressed w.r.t. $\mathcal{F}_W$, is given by the

following system of Newton-Euler equations (2.25)

$$m\ddot{\mathbf{p}} = -mg\hat{\mathbf{k}} + \mathbf{R}_{WB}\mathbf{f}_c = -mg\hat{\mathbf{k}} + \mathbf{R}_{WB}\mathbf{F}\mathbf{u} \tag{2.25a}$$

$$\mathbf{J}\dot{\boldsymbol{\omega}}_B = -\boldsymbol{\omega}_B \times \mathbf{J}\boldsymbol{\omega}_B + \boldsymbol{\tau}_c = -\boldsymbol{\omega}_B \times \mathbf{J}\boldsymbol{\omega}_B + \mathbf{M}\mathbf{u} \tag{2.25b}$$

where $g > 0, m > 0 \in \mathbb{R}$ and $\mathbf{J} \in \mathbb{R}^{3\times3}$ are respectively the gravitational acceleration, the mass of the quadrotor and its positive definite inertia matrix.

Then combining both kinematic (2.18) and dynamic (2.25) models the final mathematical model of the quadrotor is given by

$$\dot{\mathbf{p}} = \mathbf{v} \tag{2.26a}$$

$$\dot{\mathbf{R}}_{WB} = \mathbf{R}_{WB}[\boldsymbol{\omega}_B]_\times \tag{2.26b}$$

$$\dot{\mathbf{v}} = -g\hat{\mathbf{k}} + \frac{1}{m}\mathbf{R}_{WB}\mathbf{F}\mathbf{u} \tag{2.26c}$$

$$\dot{\boldsymbol{\omega}}_B = \mathbf{J}^{-1}(-\boldsymbol{\omega}_B \times \mathbf{J}\boldsymbol{\omega}_B + \mathbf{M}\mathbf{u}) \tag{2.26d}$$

Since part of the control scheme illustrated in the next chapters uses an equivalent representation of the dynamic model based on quaternion rotation it is useful to report it here

$$\dot{\mathbf{p}} = \mathbf{v} \tag{2.27a}$$

$$\dot{\mathbf{q}}_{WB} = \frac{1}{2}\mathbf{q}_{WB} \circ \widehat{\boldsymbol{\omega}_B} = \frac{1}{2}\mathbf{M}(\mathbf{q}_{WB}) \begin{bmatrix} 0 \\ \\ \boldsymbol{\omega}_B \end{bmatrix} \tag{2.27b}$$

$$\dot{\mathbf{v}} = \ddot{\mathbf{p}} = -g\hat{\mathbf{k}} + \frac{1}{m}\mathbf{R}(\mathbf{q}_{WB})\mathbf{F}\mathbf{u} \tag{2.27c}$$

$$\dot{\boldsymbol{\omega}}_B = \mathbf{J}^{-1}(-\boldsymbol{\omega}_B \times \mathbf{J}\boldsymbol{\omega}_B + \mathbf{M}\mathbf{u}) \tag{2.27d}$$

where (2.27b) is derived in A.2, equation (B.19).

It is interesting to note that the system is underactuated since to have full actuation each of the 6 DOFs must be accessible through the input but

$$rank \begin{bmatrix} m^{-1}\mathbf{R}_{WB}\mathbf{F} \\ \\ \mathbf{J}^{-1}\mathbf{M} \end{bmatrix} = rank \left(\begin{bmatrix} m^{-1}\mathbf{R}_{WB} & \mathbf{0}_{3\times1} \\ \\ \mathbf{0}_{1\times3} & \mathbf{J}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{F} \\ \\ \mathbf{M} \end{bmatrix} \right) = rank \begin{bmatrix} \mathbf{F} \\ \\ \mathbf{M} \end{bmatrix} = 4 < 6 \tag{2.28}$$

In this case since $rank(\mathbf{M}) = 3$ the rotational DOFs are all controllable independently, while $rank(\mathbf{F}) = 1$ meaning that only the body-z component of the translational DOFs can be controlled. To create an acceleration in the world-x and world-y directions there must be a change in the orientation of the quadrotor so that the direction of body-z changes, pointing towards the direction of the desired world-frame acceleration. Moreover, some of the dynamics of the system are partially coupled, as can be seen from (2.26), equation (2.26b) influences (2.26c).

3 Model Predictive Control

Model Predictive Control (MPC) is a set of control methods that, for each time instant, use the estimation of the state of a system at that moment, together with its mathematical model to predict its evolution over a finite time horizon, optimizing the potentially constrained control output to achieve a given goal.

In this chapter are given brief explanations of the concepts of optimal control and Receding Horizon Control (RHC) together with their respective problem formulations, then the specific application of NMPC on UAVs is detailed.

3.1 Background

One of the greatest results in control theory during the 1960s is the birth of the Linear Quadratic Regulator (LQR) which represented the optimal state feedback controller for a linear time-invariant Multiple-Input Multiple-Output (MIMO) system using a quadratic performance measure. It was very popular due to some properties: guarantee of existence, uniqueness and asymptotic stability under simple assumptions and finally easy to compute an explicit formula for the state feedback gain matrix.

In this section, the main ideas behind this type of control are presented since they are the basis on which the RHC principle and later its implementations MPC/NMPC were created.

Let equation (3.1) represent a generic continuous time dynamical system described by the following equation

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{x}(0) &= \mathbf{x}_0 \end{aligned} \tag{3.1}$$

where $\mathbf{x}, \mathbf{x}_0 \in \mathbb{R}^n$, $\mathbf{u} \in \mathbb{R}^m$ are respectively the state, initial state and the input of the system. The objective of infinite horizon optimal control is to find the sequence of $\mathbf{u}^*(t)$ with $t \in [0, +\infty[$ Such That (s.t.) the following cost function (3.2), defined

on the same infinite interval, is minimized.

$$J_\infty(\mathbf{x}, \mathbf{u}) = \int_0^\infty V(\mathbf{x}(t), \mathbf{u}(t))\, dt \tag{3.2}$$

Although the problem is not easy to solve in general, it is possible to prove that if $V(\cdot)$ is positive definite and both $\mathbf{f}(\cdot)$ and $V(\cdot)$ are regular enough, then $\mathbf{u}^*(t)$, the control output that minimizes (3.2), stabilizes the origin of the system for initial conditions in a neighborhood of $\mathbf{x}_0$ [11].

3.1.1 Linear Quadratic Regulator

In particular in the case of the LQR, as the name suggests, for a linear time-invariant system, characterized by the matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times m}$ the minimization problem takes the following form

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & \int_0^\infty \mathbf{x}^\top(t)\mathbf{Q}\mathbf{x}(t) + \mathbf{u}^\top(t)\mathbf{R}\mathbf{u}(t)\, dt \\ \text{s.t.} \quad & \dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ & \mathbf{x}(0) = \mathbf{x}_0 \end{aligned} \tag{3.3}$$

where the symmetric matrices $\mathbf{Q} \succeq \mathbf{0} \in \mathbb{R}^n$ and $\mathbf{R} \succ \mathbf{0} \in \mathbb{R}^m$ are respectively the weight matrices assigned to state and control output costs.
Let $\mathbf{Q}^{\frac{1}{2}} \in \mathbb{R}^n$ be a matrix s.t. $\mathbf{Q} = \mathbf{Q}^{\frac{1}{2}\top}\mathbf{Q}^{\frac{1}{2}}$, if and only if $(\mathbf{A}, \mathbf{B})$ is stabilizable and $(\mathbf{A}, \mathbf{Q}^{\frac{1}{2}})$ is detectable it holds that the Algebraic Riccati Equation (ARE)

$$\mathbf{A}^\top\mathbf{P} + \mathbf{Q} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^\top\mathbf{P} = \mathbf{0} \tag{3.4}$$

Has a unique solution $\mathbf{P}_\infty \succeq \mathbf{0}$ and if the pair $(\mathbf{A}, \mathbf{Q}^{\frac{1}{2}})$ is observable then $\mathbf{P}_\infty \succ \mathbf{0}$.
The static state feedback law that solves the problem (3.3) and makes the system asymptotically stable can be then found as

$$\mathbf{u}^*(t) = -\mathbf{K}\mathbf{x}(t) \quad \text{with} \quad \mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^\top\mathbf{P}_\infty \tag{3.5}$$

This approach has several drawbacks: it relies heavily on the accuracy of the model's dynamics, if the true evolution of the system differs ever so slightly from the predicted one the computed open-loop control law loses optimality. It does not generalize well in the context of nonlinear systems and/or cost functions, which could make the problem impossible to solve in closed form and since the solution

space is infinite-dimensional it would make it also numerically intractable. Finally even when used with linear models this type of control needs ad hoc management of state/control output constraints, which is not always easy to implement.

3.1.2 Receding Horizon Control

To tackle these problems a new approach called RHC was devised. To counteract computation difficulties, dynamical systems are considered in the discrete-time domain and the optimal control problem over a finite control horizon so that numerical approximate solutions are possible. The new problem can be formulated as

$$\begin{aligned} \min_{\mathbf{u}(\cdot)} \quad & \sum_{k=0}^{N-1} J(\mathbf{x}(k), \mathbf{u}(k)) + J_N(\mathbf{x}(N)) \\ \text{s.t.} \quad & \mathbf{x}(t+1) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \\ & \mathbf{x}(0) = \mathbf{x}(t_{start}) \\ & \mathbf{x}(k) \in \mathcal{X} \\ & \mathbf{u}(k) \in \mathcal{U} \end{aligned} \tag{3.6}$$

where $\mathbf{x}(0) = \mathbf{x}(t_{start})$ means that the initial condition of the problem is the system's condition at the beginning of each control horizon. $J(\mathbf{x}(\cdot), \mathbf{u}(\cdot))$ is the cost function at each time instant, $J_N(\mathbf{x}(N))$ is the terminal cost function and $\mathcal{X} \subseteq \mathbb{R}^n$, $\mathcal{U} \subseteq \mathbb{R}^m$ are the acceptable state and control output spaces.
The solution to this problem, Id Est, that is (i.e.) the control sequence $\mathbf{u}^*(1)$, $\mathbf{u}^*(2)$, ..., $\mathbf{u}^*(N)$ is discarded apart from the first term $\mathbf{u}^*(1)$ that is applied to the system and the problem is solved again with $t_{start} = t_{start} + 1$.

This approach basically combines finite horizon LQR control with a recursive procedure to reduce the impact of model uncertainties and introduces the possibility for numerical approximation, the main idea is summarized by Alg.1.

Algorithm 1 Receding Horizon Control.

Require: Control horizon length $N > 0$
1: **loop**
2: $\mathbf{x}_0 \leftarrow \mathbf{x}(t_{start})$
3: Re-compute optimal $\mathbf{u}^*(\cdot)$ over $[t, t+N]$ with the new $\mathbf{x}_0$
4: Apply only the first element of $\mathbf{u}^*(t)$
5: $t_{start} \leftarrow t_{start} + 1$
6: **end loop**

3.1.3 MPC

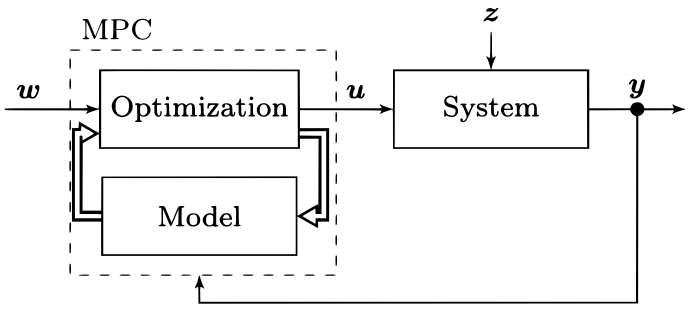


Figure 3.1: MPC scheme, taken from [4].

MPC is an implementation of RHC, in which the optimal control law is iteratively computed over the prediction horizon, online and at each time step, all while implicitly enforcing state/control output constraints. To achieve closed-loop control for each instant only the first element of the series of $\mathbf{u}^*$ is applied and the rest is discarded. The main working principle is illustrated in Fig.3.2.

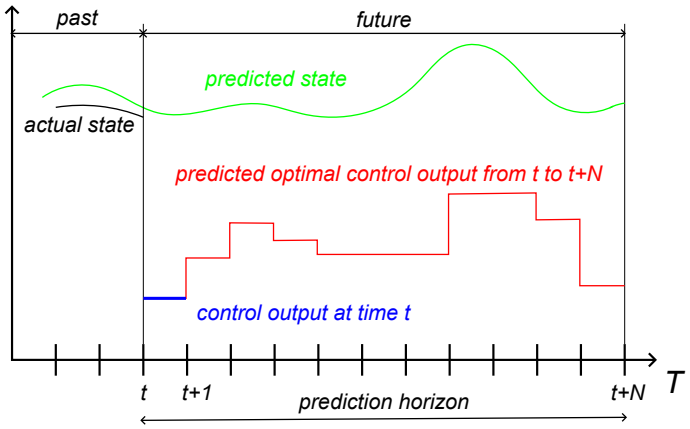


Figure 3.2: MPC working principle, adapted from [12].

This type of control has several pros: great performance, systematic constraint handling, greater attention towards the modeling of the system [4] instead of the control law and the tuning process retains a direct connection to physical parameters [5]. It also has some negative aspects, such as stability and robustness are not guaranteed, great dependence on the system model which could be inaccurate and therefore hinder the performance [7], the constraints can make the problem unfeasible at some future step [13] and finally since the optimization problem has to be solved in real-time particular attention on the processing power of the chosen hardware is needed.

NMPC

NMPC is a particular extension of the MPC framework, it is conceptually similar but it uses the nonlinear dynamic model of the system to make predictions and optimizations instead of the usual linear (or linearized) one [14], leading to significant performance improvements. Historically, due to the computational complexities, NMPC has been successfully used in industry only on systems with slow dynamics. In recent years thanks to algorithm and software optimizations, together with hardware improvements the research on its application to faster, highly nonlinear systems has been vibrant [15].

3.2 NMPC of a quad-rotor platform

Quad-rotors are a perfect, albeit challenging platform to test the limitations of a NMPC scheme since they are highly nonlinear, underactuated and have important bounds on motor output, so much so that classical control schemes based on models linearized around some working condition (like hovering) often fail to fully exploit their capabilities [2][3].

As can be seen in (3.6), three main ingredients are necessary to define a RHC problem: the discretized system, the cost and terminal cost functions, the constraints on states and control outputs and the length of the prediction horizon. These objects are defined in order below.

3.2.1 System discretization

To discretize the quad-rotor dynamical system (2.27), due to its high nonlinearity, the widely adopted explicit Runge-Kutta algorithm is applied since with it, accurate high-order numerical approximations of the functions can be constructed.

To explain its basic working principle first some preliminaries must be given: let the following be a first-order initial value problem

$$\begin{cases} \dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}(t)), \quad a \leq t \leq b \\ \mathbf{y}(t_0) = \mathbf{y}_0 \end{cases} \tag{3.7}$$

It holds by the mean value theorem for integrals that if $\mathbf{f}$(t, $\mathbf{y}$(t)) is continuous over a generic closed interval $[c, e]$ then there is at least one point $d \in [c, e]$ such that $(e - c)f(d) = \int_c^e \mathbf{f}(t, \mathbf{y}(t))\, dt$. With the same conditions, it also holds by the Fundamental Theorem of Calculus that $\mathbf{y}(e) - \mathbf{y}(c) = \int_c^e \mathbf{f}(t, \mathbf{y}(t))\, dt$.
Now if the interval $[a, b]$ is divided in N sub-intervals $[t_n, t_{n+1}]$ where $n = 0, 1, ..., N - 1$ of fixed amplitude h, one gets that

$$\mathbf{y}(t_{n+1}) - \mathbf{y}(t_n) = \int_{t_n}^{t_{n+1}} \mathbf{f}(t, \mathbf{y}(t))\, dt = h\mathbf{f}(\epsilon, \mathbf{y}(\epsilon)) \tag{3.8}$$

where $\epsilon \in [t_n, t_{n+1}]$. The function $\mathbf{f}(\epsilon, \mathbf{y}(\epsilon))$ can be approximated by the linear combinations of the evaluation of its expression for m different values of ϵ so that

the general expression of the explicit Runge-Kutta approximation for $\mathbf{y}(t_{n+1})$ is

$$\mathbf{y}(t_{n+1}) = \mathbf{y}(t_n) + h \sum_{i=1}^{m} c_i \mathbf{f}(\epsilon_i, \mathbf{y}(\epsilon_i)) \tag{3.9}$$

where different values for m and c_i are used to obtain higher or lower order approximations. In this work the fourth order one is considered and its general expression is given below

$$\begin{cases} \mathbf{y}(t_{n+1}) = \mathbf{y}(t_n) + \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \\ \mathbf{k}_1 = h\mathbf{f}(t_n, \mathbf{y}(t_n)) \\ \mathbf{k}_2 = h\mathbf{f}(t_n + \frac{1}{2}h, \mathbf{y}(t_n) + \frac{1}{2}\mathbf{k}_1) \\ \mathbf{k}_3 = h\mathbf{f}(t_n + \frac{1}{2}h, \mathbf{y}(t_n) + \frac{1}{2}\mathbf{k}_2) \\ \mathbf{k}_4 = h\mathbf{f}(t_n + h, \mathbf{y}(t_n) + \mathbf{k}_3) \end{cases} \tag{3.10}$$

For more details refer to [16].

So the system (2.27) is considered in the following in its discretized form

$$\boldsymbol{\xi}(k+1) = \mathbf{f}_k\left(\boldsymbol{\xi}(k), \mathbf{u}(k)\right) \tag{3.11}$$

where $\boldsymbol{\xi}(\cdot) \in \mathbb{R}^{13}$ is the state of the system (3 components for the position, 4 for the orientation quaternion, 3 for speed and 3 for angular speed) and $k = nT_s$, $n \in \mathbb{N}$ is the time step and $T_s > 0 \in \mathbb{R}$ is the length of the time sample. Great attention must be paid to this parameter since a value too big could lead to precision loss and thus the new system could fail to represent the whole dynamics. A value too small on the other hand leads to a heavy computational burden and thus the possible violation of the real-time constraint.

3.2.2 Cost function

A crucial part of defining a cost function is the design of the state error term

$$\mathbf{e}(k) = \begin{bmatrix} \mathbf{e}_p(k) \\ \mathbf{e}_q(k) \\ \mathbf{e}_v(k) \\ \mathbf{e}_\omega(k) \\ \mathbf{e}_\mathbf{u}(k) \end{bmatrix} \tag{3.12}$$

where $\mathbf{e}_p(k)$, $\mathbf{e}_v(k)$, $\mathbf{e}_\omega(k) \in \mathbb{R}^3$, $\mathbf{e}_\mathbf{u}(k) \in \mathbb{R}^4$ are simply the difference between position, velocity, angular velocity states, control output and the respective references at time step k. The definition of $\mathbf{e}_q(k)$ is a bit more involved since a subtraction between quaternions would not make immediate physical sense.

The distance between quaternions can be measured by different metrics, such as the Frobenius or chordal metric (3.13a) and the Riemannian or geodesic one. In this case the geodesic metric is replaced by the equivalent deviation from the identity metric (3.13b) for better computational efficiency [17]. Apart from the actual formulas the main conceptual difference between the two, considering rotation matrices $\mathbf{R}$, $\mathbf{R}_{ref} \in \mathbb{SO}(3)$, is that the first measures the length of the minimum curve that connects $\mathbf{R}$ to $\mathbf{R}_{ref}$ while the second measures the same length but the curve is restricted to belong to $\mathbb{SO}(3)$. A better visual example is reported in Fig.3.3. It is outside the scope of this work to give a more detailed look at rotation distance metrics but, since in the following only quaternions are used, it is important to note that in this context the Frobenius metric has a different interpretation, it does not represent the chord but rather the scaled-down measure of the arc length that connects $\mathbf{q}$ and $\mathbf{q}_{ref}$.

$$d_F(\mathbf{q}, \mathbf{q}_{ref}) = \sqrt{\min\left((1 - \mathbf{q} \cdot \mathbf{q}_{ref}), (1 + \mathbf{q} \cdot \mathbf{q}_{ref})\right)} \tag{3.13a}$$

$$d_I(\mathbf{q}, \mathbf{q}_{ref}) = d_R(\mathbf{q}_\mathbb{1}, \mathbf{q}^{-1} \circ \mathbf{q}_{ref}) = 2cos^{-1}\left(Re(\mathbf{q}^{-1} \circ \mathbf{q}_{ref})\right) \tag{3.13b}$$

Where $\mathbf{q}_{\mathbb{1}} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^\top$ is the quaternion that expresses a null rotation.

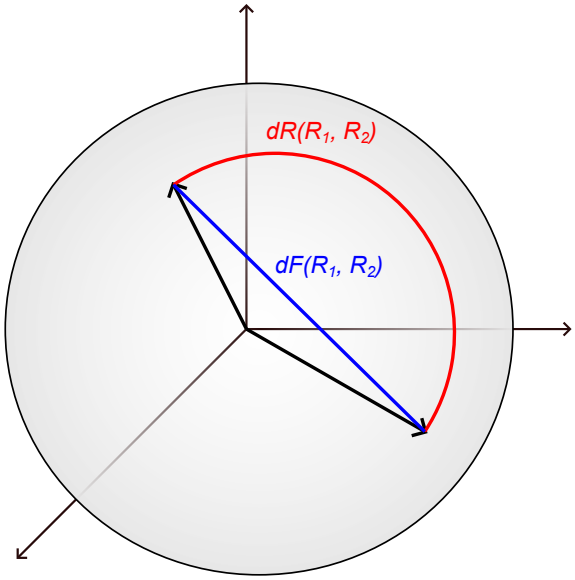


Figure 3.3: Rotation matrices metrics: Frobenius (chordal) in blue and Rieamani-ann (geodesic) in red.

To retain more information on the direction of the axis of rotation of the quadrotor w.r.t. the reference, instead of just the scalar value for distance $d_I(\mathbf{q}, \mathbf{q}_{ref})$, $\mathbf{e}_q(k)$ is defined as

$$\mathbf{e}_q(k) = Im\left(\mathbf{q}(k) \circ \mathbf{q}_{ref}^{-1}(k)\right) = Im\left(\mathbf{q} \circ \begin{bmatrix} \eta_{ref}(k) \\ -\boldsymbol{\epsilon}_{ref}(k) \end{bmatrix}\right) \tag{3.14}$$

So that the error vector can be defined as

$$\mathbf{e}(k) = \begin{bmatrix} \mathbf{p}(k) - \mathbf{p}_{ref}(k) \\ Im\left(\mathbf{q} \circ \mathbf{q}_{ref}^{-1}\right) \\ \mathbf{v}(k) - \mathbf{v}_{ref}(k) \\ \boldsymbol{\omega}(k) - \boldsymbol{\omega}_{ref}(k) \\ \mathbf{u}(k) - \mathbf{u}_{ref}(k) \end{bmatrix} \in \mathbb{R}^{16} \tag{3.15}$$

The chosen cost function for this problem is quadratic and has form

$$J(\mathbf{e}(k)) = \mathbf{e}^{\top}(k)\mathbf{Q}\mathbf{e}(k) \tag{3.16}$$

where $Q > 0 \in \mathbb{R}^{16\times16}$ is the state and control output error weight diagonal matrix. Its entries quantities mediate the relative importance of each state and control output error term in the cost function and require extensive tuning to achieve satisfactory performance.
The terminal error vector $\mathbf{e}_N$, i.e. the error vector considered at the last time step of the prediction horizon, is defined similarly to the normal one but it ignores the final control output error

$$\mathbf{e}_N = \begin{bmatrix} \mathbf{p}(N) - \mathbf{p}_{ref}(N) \\ Im\left(\mathbf{q} \circ \mathbf{q}_{ref}^{-1}\right) \\ \mathbf{v}(N) - \mathbf{v}_{ref}(N) \\ \boldsymbol{\omega}(N) - \boldsymbol{\omega}_{ref}(N) \end{bmatrix} \in \mathbb{R}^{12} \tag{3.17}$$

Thus the terminal cost function is similar to the normal one and is defined as

$$J(\mathbf{e}_N) = \mathbf{e}_N^{\top}\mathbf{Q}_N\mathbf{e}_N \tag{3.18}$$

where $\mathbf{Q}_N \in \mathbb{R}^{12\times12}$ is the terminal state error weight diagonal matrix.

3.2.3 Constraints and prediction horizon

As every actuator has limits on its outputs it is very important, especially in the case of systems with fast dynamics such as quad-rotors, to correctly estimate and set them as bounds on the control output of every control scheme. In the case of MPC/NMPC schemes one of the key advantages is explicit constraint handling of not only control outputs but also of states and it is sufficient to specify which ones are limited and their respective lower and upper bounds.

As in [18] some limits on the orientation of the quadrotor are imposed to improve stability because otherwise some dangerous maneuvers (such as 180° flips) could be commanded due to the optimization process. The orientation

bounds are given only on roll and pitch angles, since in the case of a rotor failure it is impossible to fully control the attitude of the vehicle [3]. To simplify notation in the following paragraphs time dependence on θ, ϕ, $\mathbf{q}$ and $\mathbf{u}$ is omitted.

The chosen constraints are in the form

$$\begin{cases} -\alpha \leq \theta \leq \alpha \\ -\beta \leq \phi \leq \beta \end{cases} \tag{3.19}$$

These bounds on the roll and pitch angles must be translated in terms of quaternions to be correctly imposed on the state variable $\mathbf{q}$: the elementary rotation quaternions around x (roll) and y (pitch) of α and β are given by

$$\mathbf{q}_x = \begin{bmatrix} cos(\frac{\alpha}{2}) \\ \pm sin(\frac{\alpha}{2}) \\ 0 \\ 0 \end{bmatrix} \tag{3.20a}$$

$$\mathbf{q}_y = \begin{bmatrix} cos(\frac{\beta}{2}) \\ 0 \\ \pm sin(\frac{\beta}{2}) \\ 0 \end{bmatrix} \tag{3.20b}$$

And thus the constraints can be imposed respectively on the second and third components of $\mathbf{q} = \begin{bmatrix} \eta & \epsilon_x & \epsilon_y & \epsilon_z \end{bmatrix}^\top$ so that

$$\begin{cases} -sin(\frac{\alpha}{2}) \leq \epsilon_x \leq sin(\frac{\alpha}{2}) \\ -sin(\frac{\beta}{2}) \leq \epsilon_y \leq sin(\frac{\beta}{2}) \end{cases} \tag{3.21}$$

The constraints on the thrust commanded to the motor are straightforward

$$0 \leq \mathbf{u} \leq \mathbf{u}_{max} \tag{3.22}$$

where $\mathbf{u}_{max}$ is the maximum output thrust vector that contains the maximum thrusts of each motor.

The final key parameter to consider is the number of steps in the prediction horizon N, since the system is discretized using a constant time step size of T_s seconds one has

$$N = \frac{T_h}{T_s} \tag{3.23}$$

where T_h is the length in seconds of the prediction horizon. This parameter must be tuned correctly to have a good trade-off between computational burden and the capability of predicting enough evolution of the dynamics of the system to understand if a control output is good or not.

3.2.4 Final problem formulation

By combining (3.11), (3.16), (3.18), (3.21), (3.22) and (3.23) the final formulation of the NMPC optimization problem applied to a quadrotor platform can be defined

$$\begin{aligned}
\min_{\mathbf{u}(\cdot)} \quad & \sum_{k=0}^{N-1} \left(\mathbf{e}^\top(k)\mathbf{Q}\mathbf{e}(k)\right) + \mathbf{e}_N^\top \mathbf{Q}_N \mathbf{e}_N \\
\text{s.t.} \quad & \xi(k+1) = \mathbf{f}_k\left(\xi(k), \mathbf{u}(k)\right) \\
& \xi(0) = \xi(t) \\
& |\epsilon_x| \leq sin(\frac{\alpha}{2}) \\
& |\epsilon_y| \leq sin(\frac{\beta}{2}) \\
& 0 \leq \mathbf{u} \leq \mathbf{u}_{max}
\end{aligned} \tag{3.24}$$

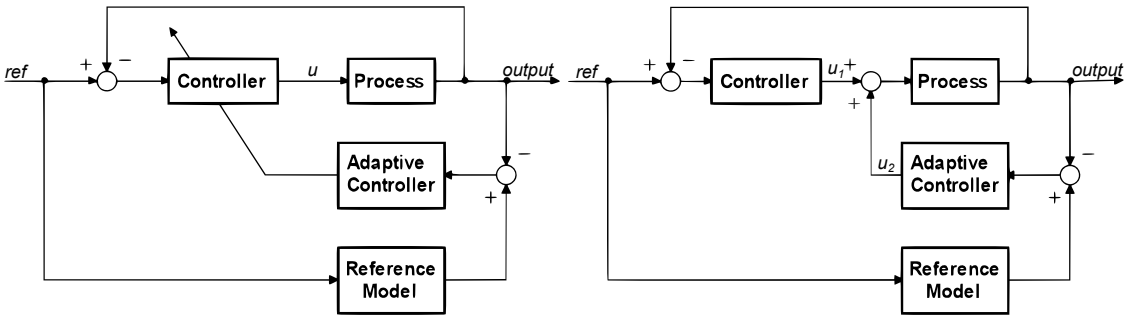
All the numerical values of the parameters are given in the results Chp.6.

4 Adaptive control

Since MPC schemes performance degrades without an accurate model of the system, which is often too difficult or costly to estimate [7], in [19] the planning advantage of NMPC is augmented via control output addition with a purely reactive adaptive control scheme called $\mathcal{L}_1$. In this chapter its formulation is derived and explained together with the basic principle of MRAC.

4.1 Background

MRAC is a set of control methods that, for each time instant, use the mathematical model of a system to characterize the desired closed-loop performance. When it strays from the ideal one, depending on the magnitude of the difference, some modifications to the parameters in the main controller are introduced (Fig.4.1a) or an extra control output is provided (Fig.4.1b).



(a) (b)

Figure 4.1: MRAC block schemes, adapted from [20].

This type of control although very useful has some drawbacks: it is sensitive to adaptive gains, which proved difficult to tune experimentally, resulting in high-frequency oscillations in the control output, large transient errors or slow convergence rates[21].

For these reasons, a new control architecture has been developed called $\mathcal{L}_1$. It

brings some important advantages w.r.t. the classical MRAC such as easy tuning and reduction of the control output high-frequency components. The main differences between the two are that in MRAC the control signals depend directly on parameter estimates, while in $\mathcal{L}_1$ they are also filtered by a low pass filter that cuts off the frequencies that could excite unstable modes, regulating the trade-off between robustness and adaptation. This means that large gains are more easily tunable by accurately selecting the filter. The $\mathcal{L}_1$ also uses state predictions which provide the reference to follow when the tracking/estimation errors are small enough. Its name derives from the fact that the bounds on the adaptation errors depend on the adaptation gains, on the largest values of the unknown parameters and on the L1 norm of the filter's transfer function [22].

Due to the nature of NMPC, which plans according to a model to achieve optimal tracking, $\mathcal{L}_1$ represents a great addition to the control scheme since it purely reacts to model uncertainties trying to compensate for them while accounting for the dynamics of the system [8]. By combining the two one gets a system that is capable of long-term planning and fast adaptation to disturbances and modeling errors.

4.2 $\mathcal{L}_1$ CONTROL OF A QUAD-ROTOR PLATFORM

Even though quad-rotors are becoming more and more common in the consumer and industrial markets huge variability in components and manufacturing tolerances make it very hard to obtain precise physical characteristics from the data sheets of the parts, which often are not even available. Moreover, they are usually designed to survive crashes and ungraceful landings but there could be deformations in the parts that can impact the aerodynamic and inertial properties of the frame and propellers [6]. Also external disturbances like wind are difficult to predict in real-time, therefore the inclusion of an adaptive control contribution when dealing with small UAVs like quad-rotors is very important to compensate for unaccounted model mismatches and environmental challenges without prior assumptions, while pushing the systems to their dynamical limits.

The control law formulation starts by manipulating a bit the dynamic model of the quadrotor (2.25) and adding some terms that contain the uncertainties. Note that to reduce clutter in the notation time dependence is omitted but it is important to keep in mind that the pose, forces, torques and uncertainties are all

time-varying.

$$\dot{\mathbf{v}} = -g\hat{\mathbf{k}} + \frac{1}{m}\mathbf{R}_{WB}\mathbf{f}_c + \frac{1}{m}\mathbf{R}_{WB}\mathbf{f}_c\boldsymbol{\delta} \tag{4.1a}$$

$$\dot{\boldsymbol{\omega}}_B = \mathbf{J}^{-1}\left(-\boldsymbol{\omega}_B \times \mathbf{J}\boldsymbol{\omega}_B + \boldsymbol{\tau}_c + \boldsymbol{\varrho}\right) \tag{4.1b}$$

Where $\boldsymbol{\delta} = \begin{bmatrix} \delta_x & \delta_y & \delta_z \end{bmatrix}^\top \in \mathbb{R}^3$, $\boldsymbol{\varrho} = \begin{bmatrix} \varrho_x & \varrho_y & \varrho_z \end{bmatrix}^\top \in \mathbb{R}^3$ are respectively the uncertainties appearing in the linear and angular accelerations. Remember also that $\mathbf{R}_{WB}$ can be found from the quaternion representation of pose using Rodrigues formula (2.16).

Since a quad-rotor is underactuated (see the end of chapter 2), only the z component of $\boldsymbol{\delta}$ and all of $\boldsymbol{\varrho}$ can be directly compensated for and thus the matched uncertainty vector (i.e. the vector that contains the uncertainties that can be compensated directly through the controlled input of the system) is defined as $\boldsymbol{\sigma}_m = \begin{bmatrix} \delta_z & \varrho_x & \varrho_y & \varrho_z \end{bmatrix}^\top$. The uncertainties in acceleration in the x and y directions cannot be directly countered, therefore they make up the unmatched uncertainty vector $\boldsymbol{\sigma}_{um} = \begin{bmatrix} \delta_x & \delta_y \end{bmatrix}^\top$.

Now consider a reduced state variable $\mathbf{z} = \begin{bmatrix} \mathbf{v}^\top & \boldsymbol{\omega}_B^\top \end{bmatrix}^\top \in \mathbb{R}^6$, its derivative can be rewritten as

$$\dot{\mathbf{z}} = \mathbf{f}(\mathbf{R}_{WB}) + \mathbf{g}(\mathbf{R}_{WB})(\mathbf{u}_{\mathcal{L}_1} + \boldsymbol{\sigma}_m) + \mathbf{g}^{\perp}(\mathbf{R}_{WB})\boldsymbol{\sigma}_{um} \tag{4.2}$$

where $\mathbf{u}_{\mathcal{L}_1}$ is the control output of the $\mathcal{L}_1$ controller, $\mathbf{f}(\mathbf{R}_{WB}) \in \mathbb{R}^6$ represents the desired dynamics, $\mathbf{g}(\mathbf{R}_{WB}) \in \mathbb{R}^{6\times4}$ the uncertainty in the matched component of the dynamics and $\mathbf{g}^{\perp}(\mathbf{R}_{WB}) \in \mathbb{R}^{6\times4}$ the unmatched one. They are defined as

$$\mathbf{f}(\mathbf{R}_{WB}) = \begin{bmatrix} -g\hat{\mathbf{k}} + \frac{1}{m}\mathbf{R}_{WB}\mathbf{f}_c \\ \mathbf{J}^{-1}\left(-\boldsymbol{\omega}_B \times \mathbf{J} + \boldsymbol{\tau}_c\right) \end{bmatrix} \tag{4.3a}$$

$$\mathbf{g}(\mathbf{R}_{WB}) = \begin{bmatrix} \frac{\hat{\mathbf{k}}}{m}\mathbb{1}_{1\times4} \\ \mathbf{J}^{-1}\mathbf{M} \end{bmatrix} \tag{4.3b}$$

$$\mathbf{g}^{\perp}(\mathbf{R}_{WB}) = \begin{bmatrix} \frac{\hat{\mathbf{i}}}{m} & \frac{\hat{\mathbf{j}}}{m} \\ \\ \mathbb{0}_{3\times1} & \mathbb{0}_{3\times1} \end{bmatrix} \tag{4.3c}$$

where $\mathbf{f}_c = \mathbf{F}\mathbf{u}_{NMPC}$ and $\boldsymbol{\tau}_c = \mathbf{M}\mathbf{u}_{NMPC}$ are the thrust and torque defined in (2.23) generated by the NMPC control output, $\mathbb{1}_{1\times4}$ is a row vector of ones with four columns, $\mathbb{0}_{3\times1}$ is a column vector of zeros with three rows and $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$ and $\hat{\mathbf{k}}$ are the usual canonical versors of the world reference frame.

The $\mathcal{L}_1$ observer is defined as

$$\dot{\widehat{\mathbf{z}}} = \mathbf{f}(\mathbf{R}_{WB}) + \mathbf{g}(\mathbf{R}_{WB})(\mathbf{u}_{\mathcal{L}_1} + \hat{\boldsymbol{\sigma}}_m) + \mathbf{g}^{\perp}(\mathbf{R}_{WB})\hat{\boldsymbol{\sigma}}_{um} + \mathbf{A}_s\widetilde{\mathbf{z}} \tag{4.4}$$

where $\widehat{\mathbf{z}}$ is the state predicted by the $\mathcal{L}_1$ observer, $\mathbf{z}$ is the state obtained by an estimator that outputs the state of the quad-copter (some implementation details on this are given in sec.5.1) and $\widetilde{\mathbf{z}} = \widehat{\mathbf{z}} - \mathbf{z}$. The derivative of the observer error can thus be found as

$$\begin{aligned} \dot{\widetilde{\mathbf{z}}} &= \mathbf{A}_s\tilde{\mathbf{z}} + \begin{bmatrix} \mathbf{g}(\mathbf{R}_{WB}) & \mathbf{g}^{\perp}(\mathbf{R}_{WB}) \end{bmatrix} \left(\begin{bmatrix} \widehat{\boldsymbol{\sigma}}_m \\ \\ \widehat{\boldsymbol{\sigma}}_{um} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\sigma}_m \\ \\ \boldsymbol{\sigma}_{um} \end{bmatrix} \right) \\ &= \mathbf{A}_s\tilde{\mathbf{z}} + \mathbf{G}(\mathbf{R}_{WB})(\widehat{\boldsymbol{\sigma}} - \boldsymbol{\sigma}) \end{aligned} \tag{4.5}$$

The analytical solution of this differential equation is the usual convolution

$$\widetilde{\mathbf{z}}(t) = e^{\mathbf{A}_s t}\widetilde{\mathbf{z}}(0) + \int_0^t e^{\mathbf{A}_s(t-\tau)}\mathbf{G}(\mathbf{R}_{WB}(\tau))[\widehat{\boldsymbol{\sigma}}(\tau) - \boldsymbol{\sigma}(\tau)]\, d\tau \tag{4.6}$$

Here only the main points of this demonstration are summarized, for the full derivation refer to [23].

To discretize this equation let $\widetilde{\mathbf{z}}(k) = \widetilde{\mathbf{z}}(nT_s)$ where $n \in \mathbb{N}$ and $T_s \in \mathbb{R}^+$ is the length of the time step. it follows that

$$\begin{aligned} \widetilde{\mathbf{z}}(k) &= e^{\mathbf{A}_s kT_s}\widetilde{\mathbf{z}}(0) + \int_0^{kT_s} e^{\mathbf{A}_s(kT_s-\tau)}\mathbf{G}(\mathbf{R}_{WB}(\tau))[\widehat{\boldsymbol{\sigma}}(\tau) - \boldsymbol{\sigma}(\tau)]\, d\tau \\ \widetilde{\mathbf{z}}(k+1) &= e^{\mathbf{A}_s(k+1)T_s}\widetilde{\mathbf{z}}(0) + \int_0^{(k+1)T_s} e^{\mathbf{A}_s((k+1)T_s-\tau)}\mathbf{G}(\mathbf{R}_{WB}(\tau))[\widehat{\boldsymbol{\sigma}}(\tau) - \boldsymbol{\sigma}(\tau)]\, d\tau \\ \widetilde{\mathbf{z}}(k+1) &= e^{\mathbf{A}_s T_s}\widetilde{\mathbf{z}}(k) + \int_{kT_s}^{(k+1)T_s} e^{\mathbf{A}_s((k+1)T_s-\tau)}\mathbf{G}(\mathbf{R}_{WB}(\tau))[\widehat{\boldsymbol{\sigma}}(\tau) - \boldsymbol{\sigma}(\tau)]\, d\tau \end{aligned} \tag{4.7}$$

Now, let $\nu(\tau) = (k+1)T_s - \tau$ and assume that $\mathbf{G}(\mathbf{R}_{WB}(\tau))[\widehat{\boldsymbol{\sigma}}(\tau) - \boldsymbol{\sigma}(\tau)]$ to be constant during the interval during a time step, (4.7) becomes

$$\begin{aligned}\widetilde{\mathbf{z}}(k+1) &= e^{\mathbf{A}_s T_s}\widetilde{\mathbf{z}}(k) - \left(\int_{\nu(kT_s)}^{\nu((k+1)T_s)} e^{\mathbf{A}_s \nu}\, d\nu\right)\mathbf{G}(\mathbf{R}_{WB}(k))[\widehat{\boldsymbol{\sigma}}(k) - \boldsymbol{\sigma}(k)] \\ &= e^{\mathbf{A}_s T_s}\widetilde{\mathbf{z}}(k) - \left(\int_{T_s}^{0} e^{\mathbf{A}_s \nu}\, d\nu\right)\mathbf{G}(\mathbf{R}_{WB}(k))[\widehat{\boldsymbol{\sigma}}(k) - \boldsymbol{\sigma}(k)] \\ &= e^{\mathbf{A}_s T_s}\widetilde{\mathbf{z}}(k) + \mathbf{A}_s^{-1}\left(e^{\mathbf{A}_s T_s} - I_{6\times 6}\right)\mathbf{G}(\mathbf{R}_{WB}(k))[\widehat{\boldsymbol{\sigma}}(k) - \boldsymbol{\sigma}(k)]\end{aligned} \tag{4.8}$$

It can be proven that the $\mathcal{L}_1$ control is stable and that all outputs, states and estimation errors remain bounded so the terms $\widetilde{\mathbf{z}}(k+1)$ and $\mathbf{A}_s^{-1}\left(e^{\mathbf{A}_s T_s} - I_{6\times 6}\right)\mathbf{G}(\mathbf{R}_{WB}(k))\boldsymbol{\sigma}(k)$ can be ignored and thus

$$-e^{\mathbf{A}_s T_s}\widetilde{\mathbf{z}}(k) = \mathbf{A}_s^{-1}\left(e^{\mathbf{A}_s T_s} - I_{6\times 6}\right)\mathbf{G}(\mathbf{R}_{WB}(k))\widehat{\boldsymbol{\sigma}}(k) \tag{4.9}$$

By inverting the last equation and grouping some terms one gets an expression for the estimated uncertainties

$$\begin{bmatrix}\widehat{\boldsymbol{\sigma}}_m(k) \\ \\ \widehat{\boldsymbol{\sigma}}_{um}(k)\end{bmatrix} = \widehat{\boldsymbol{\sigma}}(k) = -\mathbf{G}^{-1}(\mathbf{R}_{WB}(k))\boldsymbol{\Phi}^{-1}\boldsymbol{\mu} \tag{4.10}$$

where $\boldsymbol{\Phi} = \mathbf{A}_s^{-1}\left(e^{\mathbf{A}_s T_s} - I_{6\times 6}\right)$ and $\boldsymbol{\mu} = e^{\mathbf{A}_s T_s}\widetilde{\mathbf{z}}(k)$.

One of the main characteristics of the $\mathcal{L}_1$ control is the filtering that allows a trade-off between robustness and adaptation so the control law is given by

$$\mathbf{u}_{\mathcal{L}_1} = \mathbf{C}(s)\widehat{\boldsymbol{\sigma}}_m \tag{4.11}$$

where $\mathbf{C}(s)$ is a first order strictly proper continuous-time low pass filter with cutoff frequency ω_{co}.

In practice, since the control is implemented in discrete time both state estimation and control action are propagated forward in time using discretized versions of their expressions where the dependence of $\mathbf{f}$, $\mathbf{g}$ and $\mathbf{g}^{\perp}$ on $\mathbf{R}_{WB}$ is omitted to simplify notation and the k in the subscripts indicates the time step at which the quantities are calculated

$$\mathbf{u}_{\mathcal{L}_1,k} = \mathbf{u}_{\mathcal{L}_1,k-1} e^{-\omega_{co} T_s} - \widehat{\boldsymbol{\sigma}}_{m,k}(1 - e^{-\omega_{co} T_s}) \tag{4.12}$$

$$\widehat{\mathbf{z}}(k+1) = \widehat{\mathbf{z}}(k) + [\mathbf{f}_k + \mathbf{g}_k(\mathbf{u}_{\mathcal{L}_1,k} + \hat{\boldsymbol{\sigma}}_{m,k}) + \mathbf{g}_k^{\perp} \hat{\boldsymbol{\sigma}}_{um,k} + \mathbf{A}_s \widetilde{\mathbf{z}}(k)] T_s \tag{4.13}$$

5 Complete control system

This chapter provides a complete overview of the implementation of the control schemes discussed in the previous chapters. It starts with a brief summary of PX4's main features and strengths, then there is an explanation of how it is connected to MATLAB/Simulink through ROS 2™ and then finally the complete control architecture used is provided and illustrated.

5.1 PX4-Autopilot

PX4 is an open-source project that aims at providing a development environment and a flexible set of tools for drones and other unmanned vehicles research. It effectively creates a scalable ecosystem both in software and hardware. Here are reported some of its main features and strengths [24][25]:

- Modularity: PX4 can be enhanced by adding sensors, components and new software modules which do not influence the base system functionality and new features can be built without compromising stability or performance of the extended system.
- Open-source license (BSD 3-clause license): the PX4 ecosystem is being constantly developed by an active community all around the world, meaning that it is not specialized on any particular commercial solution, therefore it remains a general toolkit that can be adapted by both industry and academic research.
- Configurability: every feature is encapsulated in a self-contained module that can be changed without modifying the core codebase. This, together with the provided optimized Application Programming Interfaces (APIs) and Software Development Kits (SDKs) allow easy creation, testing and deployment of new modules.
- Autonomy Stack: PX4 is designed to provide a low barrier of entry for developers interested in autonomous and/or vision capabilities.
- Validated hardware: there are several pre-configured UAVs kits available that provide every component needed with assembly and configuration instructions to provide a mostly seamless experience and fast first setups.

- Safety: PX4 has several configurable and automated safety features already built-in such as parachute deployment, different return modes battery and actuator monitoring, etc...
- Extensive testing: since PX4 software is at the base of many commercial and non-commercial projects it is constantly tested by a lot of users with lots of different use cases that can contribute to debugging.
- Standardization: thanks to its wide user base there is a constant effort toward standardization of communication protocols, peripheral integration and power management between different platforms.
- Documentation: being a widely adopted open-source environment there is a good focus on the production of easy-to-understand and effective documentation.

The version used in this work is available on GitHub at [26].

PX4 software is mainly divided into two layers: the flight stack and the middleware. Both are briefly presented below.

5.1.1 Flight stack

The flight stack is the collection of control and guidance algorithms that allow the drones different levels of autonomy and flight modes. It includes state estimators for position and attitude, different controllers for n-copters, VTOL and fixed-wing airframes, Radio Command (RC) input processing and actuator low-level control. The main components of the flight stack are illustrated in Fig.5.1 and explained below.

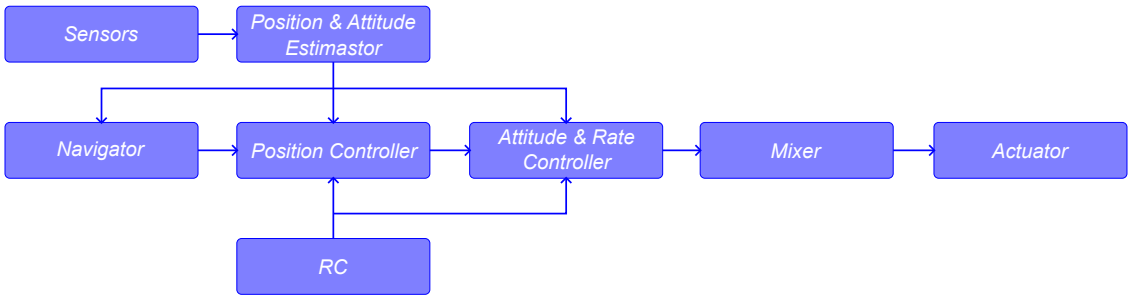


Figure 5.1: Flight stack block representation, taken from [27].

- Sensors: everything that collects data about the environment or the drone itself, such as vision systems, magnetometers, Inertial Measurement Unit (IMU), barometers, etc...

- Estimator: algorithm that takes as input sensor data from different sources, combines them and outputs an estimation of the vehicle state. Specifically PX4 uses an Extended Kalman Filter (EKF) which is a complex but very powerful algorithm, for more details see [28].
- Navigator: algorithm that sets high-level goal states for the drone depending on state estimation and general mission objectives.
- Controller: algorithm that takes as input a setpoint given by the navigator and using state estimate information produces an output for the motors to get to achieve said setpoint.
- RC: Device used to manually give setpoints to the drone
- Mixer: algorithm that takes force commands given by the controller and transforms them into individual motor commands, while checking for actuator limits. Its operation depends on the motor arrangement, vehicle type and other factors. The outputs of the mixer are sent to the actuator driver (such as Electronic Speed Controllers (ESCs)) which translates it to the right communication protocol to be understood by the actuator.
- Actuators: Devices that translate an input signal in a physical effect such as rotation and thus thrust for motors with propellers or pan and tilt for a servo-mounted camera.

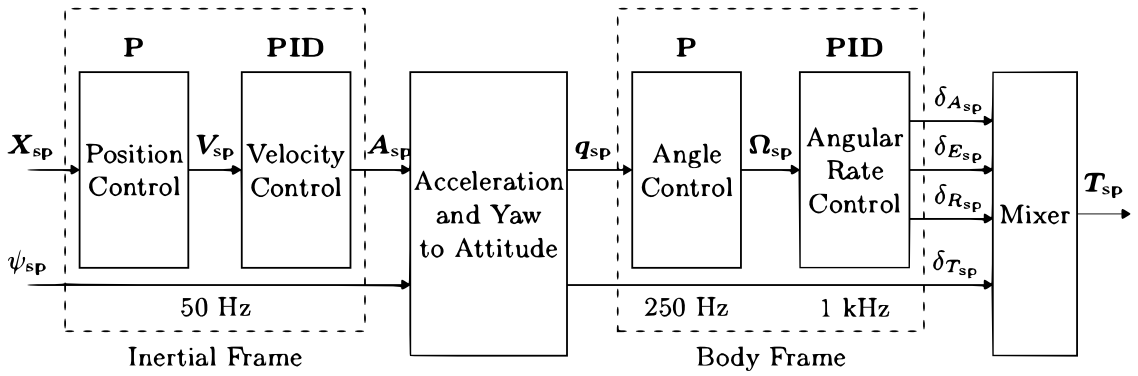


Figure 5.2: PX4 cascaded control scheme, taken from [29].

In this work the importance of the stock PX4 controllers, presented in Fig.5.2, is greatly reduced since the developed control scheme is acting directly at the motor thrust level at the mixer, effectively disabling the upstream blocks.

5.1.2 Middleware

The middleware is the code infrastructure that provides parallelized communication between every module and between the flight stack and the external world. Its main functions are summarized below

- Device drivers: the middleware houses all component drivers and is therefore responsible for their correct functioning and correct communication with the rest of the software.
- Inter-module communication: using the uORB message bus [30], an asynchronous publish-subscribe messaging API every module can safely publish some messages that some other module can subscribe to and thus read.
- External communication: using the MAVLink protocol [31], uORB messages can be sent to external software/hardware that is outside of the flight stack such as QGround Control (QGC) (more on this application in sec.5.2).
- ROS 2 communication: using the XRCE-DDS middleware uORB topics can be published/subscribed to as they were ROS 2 topics (more on this in sec.5.3).
- Simulation: the middleware provides a SITL/Hardware In the Loop (HITL) simulation layer that allows PX4 flight code to run on a simulated PX4 board on a companion computer/real PX4 hardware to control a simulated drone. This is crucial for the simulation and controller validation more on this is discussed in chapter 6.

5.2 QGroundControl©

QGC is an extremely useful application that provides easy access to the features of PX4 through an intuitive graphic interface. Its main functions are:

- Full configuration of PX4 enabled vehicles, from the shape of the frame to the motor limits, RC mappings, failsafe behavior, and much more. It allows to basically check and change every parameter of the drone's firmware.
- Access to macro commands like automated takeoff, path and mission planning/following with waypoints for autonomous flight.
- Display the flight map that shows vehicle position, flight track, waypoints and vehicle instruments.
- Video streaming from the drone's camera if it has one. Instrument displays can be overlayed on the stream.

- Download flight logs that can be later analyzed using the Flight Review web app [32].
- Access to the MAVLink console running on the vehicle to inspect messages and send commands.
- Compatibility with every major operative system, from Windows, Linux and macOS to Android and iOS.

In this work QGC is mainly used to ensure that the connection between the quad-copter and companion PC is stable and working and to tune some of PX4's parameters. For a more in-depth look refer to [33].

5.3 ROS 2™

ROS 2 or Robot Operating System 2 is a set of libraries hugely popular in robotic applications. In this work it is used merely as a communication layer between PX4 software and MATLAB/Simulink, in which the control architecture was actually implemented.

ROS 2 communicates with PX4 through the XRCE-DDS middleware, clearly illustrated in Fig.5.3, there is a client on PX4 that packages and formats uORB messages to be compatible with the ROS network and an agent on a companion PC that collects the messages published by ROS topics to be sent out. Client and agent communicate bi-directionally over a serial or UDP link.

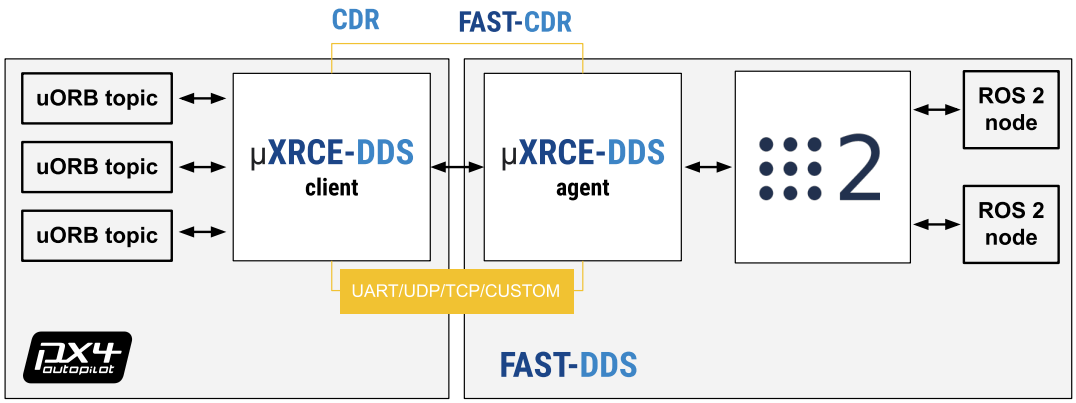


Figure 5.3: PX4-ROS 2 communication scheme, taken from [34].

5.4 Implementation details

First of all, because it is constantly developing and changing, it is important to specify that the PX4-Autopilot version used in this work is the `v1.14.0-beta1`. Since the control scheme in this thesis uses control inputs at the rotor thrust level and as state feedback the estimation provided by the PX4 Kalman estimator fed by a vision system, some minor modifications to the default PX4 software are required in order to make everything work correctly.

Starting from the Kalman filter settings, a single drone parameter has to be changed and can be easily done using QGC

```
EKF2_HGT_REF 3
```

This tells the Kalman filter that the vision data is the reference source of information for altitude estimation.

By default the part of the middleware that publishes uORB topics on the ROS 2 network (a module called `microdds_client`) is receiving and publishing to only a subset of the available topics. To access the ones needed for this work's control scheme some modifications to `PX4-Autopilot/src/modules/microdds_client/dds_topics.yaml` have to be made.
At the end of the subscriptions section add the following lines

```
- topic: /fmu/in/actuator_motors
    type: px4_msgs::msg::ActuatorMotors
```

This is needed to allow sending direct motor controls to the drone through ROS 2.
At the end of the publications section add the following lines

```
- topic: /fmu/out/vehicle_visual_odometry
    type: px4_msgs::msg::VehicleOdometry

- topic: /fmu/out/actuator_motors
    type: px4_msgs::msg::ActuatorMotors

- topic: /fmu/out/vehicle_rates_setpoint
    type: px4_msgs::msg::VehicleRatesSetpoint
```

These modifications are useful for debugging purposes. They enable the external monitoring of several important messages: outputs of the vision system before it is processed by the Kalman estimator (to verify that the final estimation is close enough to the true values) and motor commands (to verify that the ones sent are received correctly).

5.5 MATLAB®/Simulink®

The complete control structure is reported in Fig.5.4 where in blue are represented the blocks implemented in Simulink, in yellow are the ones that belong to PX4 software, in green is the vision system and in red is the drone. Instead, the line width and pattern indicate the data transmission medium, the normal ones are Simulink signals, the bold ones are communications inside the PX4 flight stack and the bold dotted ones are transmissions through the ROS 2 network. Finally, the fine dotted lines represent the physical link between quad-rotor dynamics and sensor/vision measurements.

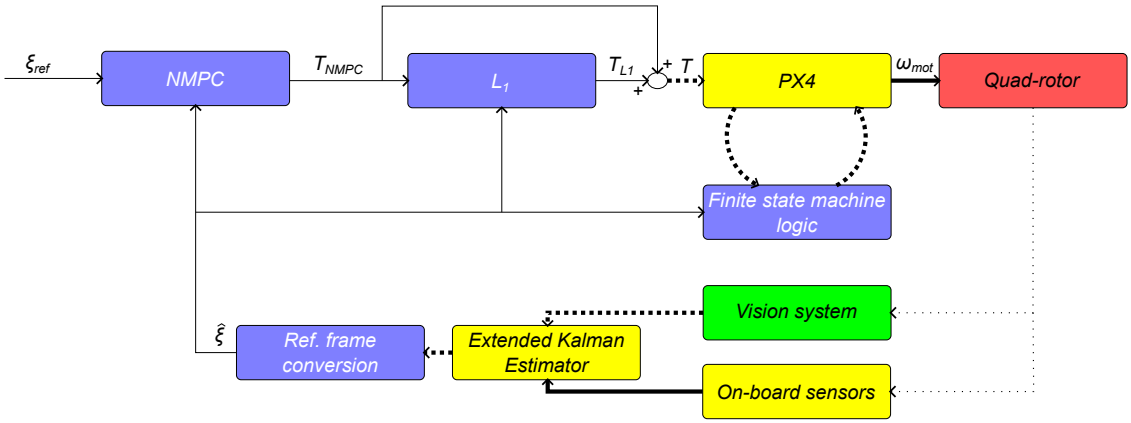


Figure 5.4: Complete control scheme used throughout this work.

Now from left to right and top to bottom the Simulink blocks are explained in more detail one at a time.

5.5.1 NMPC block

This block contains an implementation of the NMPC algorithm developed at the University of Padova called MATMPC [35]. This tool is open source and aims at providing an easy-to-use and very optimized NMPC implementation with lots of possible options (Fig.5.5). The optimal control problem is discretized by multiple shooting with support for the fixed step (explicit/implicit) Runge-Kutta method presented in sec.3.2.1 and the resulting nonlinear program is solved by Sequential Quadratic Programming (SQP) method. The derivatives that are needed to perform optimization are obtained by CasADi [36], the state-of-the-art automatic/algorithmic differentiation toolbox. All of the functions of this toolbox are written in the MATLAB API for C so that both the readability of MATLAB functions and the great performance of the C language are mostly preserved.

The model used for prediction is created with a simple MATLAB script that contains the definition of states, control outputs, error metrics, output/state constraints and dynamic equations. It is then compiled separately before the execution of the scheme. This enables fast swapping between models since once ready it is sufficient to compile the desired one.

Hessian Approximation	Gauss-Newton		
Integrator	Explicit Runge Kutta 4 (`CasADi` code generation)	Explicit Runge Kutta 4	Implicit Runge-Kutta (Gauss-Legendre)
Condensing	non	full	partial
QP solver	`qpOASES` `OSQP`	`MATLAB quadprog` `HPIPM`	`Ipopt`
Globalization	ℓ_1 merit function line search		Real-Time Iteration
Additional features	CMoN-SQP		input MB

Figure 5.5: Options and features available in MATMPC, taken from [35].

5.5.2 $\mathcal{L}_1$ BLOCK

This block receives as input the control output of the NMPC block and implements with a MATLAB functions the equations (4.8), (4.12) and (4.13) presented in sec.4.2.

5.5.3 FINITE STATE MACHINE BLOCK

This block is fundamental to this work's application, it coordinates the activities of all the ROS 2 publishers and subscribers, communicates with PX4 to check diagnostic signals and manages the flow of the program.

It receives several inputs:

- From the user, simulation settings and takeoff height.
- From PX4 message definition, constants that represent specific commands for the PX4 software (arm motors, change to offboard mode and land).
- Position and attitude feedback.
- From ROS 2 network, flags used for failsafe behavior.

And has several outputs:

- Boolean flags that set which control mode is being used depending on the execution phase.
- Signal that controls the activation of the subsystems for publishing vehicle commands, trajectory setpoints and motor commands depending on the execution phase.

- Vehicle command to be sent to the PX4 software (arm, land, etc...)
- The faultState and faultFlags variables which contain the fault state (a number different from zero, depending on the flight phase) and the status of some of the safety flags published by PX4, together with a check on the publication frequency of the vehicle odometry since frequently updated feedback is crucial when dealing with agile nonlinear systems. These output variables are stored in memory after every execution and used for debugging purposes.

Fig.5.6 illustrates the composition and connections between states of the flow chart. There are 4 distinct phases in the nominal working condition:

1. pre-flight in which PX4 is put into off-board position control mode and the motors are armed.
2. position-mode takeoff in which using a fixed position set-point reference is given to get to a user-given starting position.
3. actuator-mode tracking in which the quad-copter is controlled with direct single motor thrust commands given by the sum of the NMPC and $\mathcal{L}_1$ blocks. The objective in this phase is to follow the reference trajectory as closely as possible.
4. position-mode landing in which the control is switched back to position mode and the PX4 macro command to land is issued then, once on the ground, the motors are disarmed.

Given the potential danger of working with spinning propellers and fast aerial vehicles, some failsafes are put in place to ensure correct and safe operation. These are mainly represented by the fail flags published by PX4 since a lot of checks on hardware (on motors, batteries, sensors, etc...) and on the pose estimations are already baked into the flight stack.
Also because vision information is crucial in this application, a new flag is introduced to check on the frequency of publication of new messages on the vision subsystem topic.
In the Simulink scheme, these flags trigger different behaviors depending on the flight phase. Specifically, execution stops if the drone is still on the ground and the immediate landing command is given if it is flying.
Other nice to have redundancy checks are put in place to avoid possible blocks in execution, such as if the drone is stuck in the takeoff or in the follow trajectory states after some time set by the user the landing command is automatically given, or if PX4's landing detector doesn't recognize correctly that the drone has stopped after another timer set by the user it automatically disarms the motors.

For obvious reasons the only failure mode that cannot be addressed by Simulink is when the connection between the ground station and the drone is lost, in this case PX4 software automatically goes into a mode (hold, land, etc...) that can be set in several parameters via QGC.

As last comment, it is important to understand that since PX4 position control is just a proportional gain controller (see Fig.5.2) therefore some error in the initial condition, before tracking the reference, is to be expected. The function that tells if the drone is at the desired height is just used to check if the height and yaw angle reported by the vision topic are inside fairly large acceptable ranges ($\pm 30cm$).

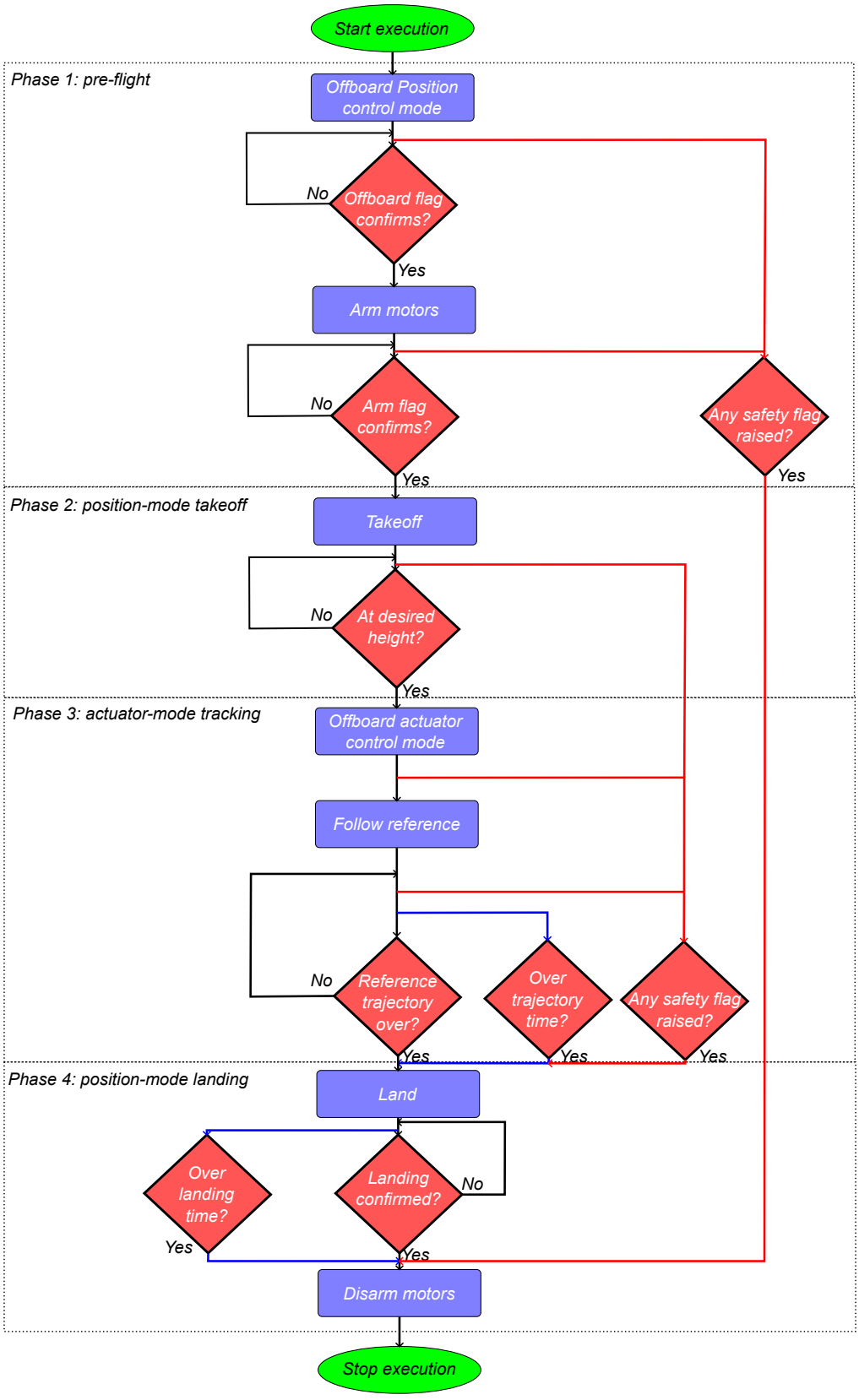


Figure 5.6: Simplified flow chart of the flight logic. The red connections indicate safety behavior, the blue ones redundancy behavior and the black ones the normal flow of the program.

5.5.4 Reference frame conversion block

PX4 and the classic dynamical models for quad-rotors, such as the one derived here, use different conventions to define their reference frames, both body and world-fixed and are presented in Fig.5.7. In the PX4 context, FRD for the body frame and NED/FRD for the fixed frame and in MATLAB/ROS 2 context FLU for the body frame and FLU/ENU for the fixed frame.

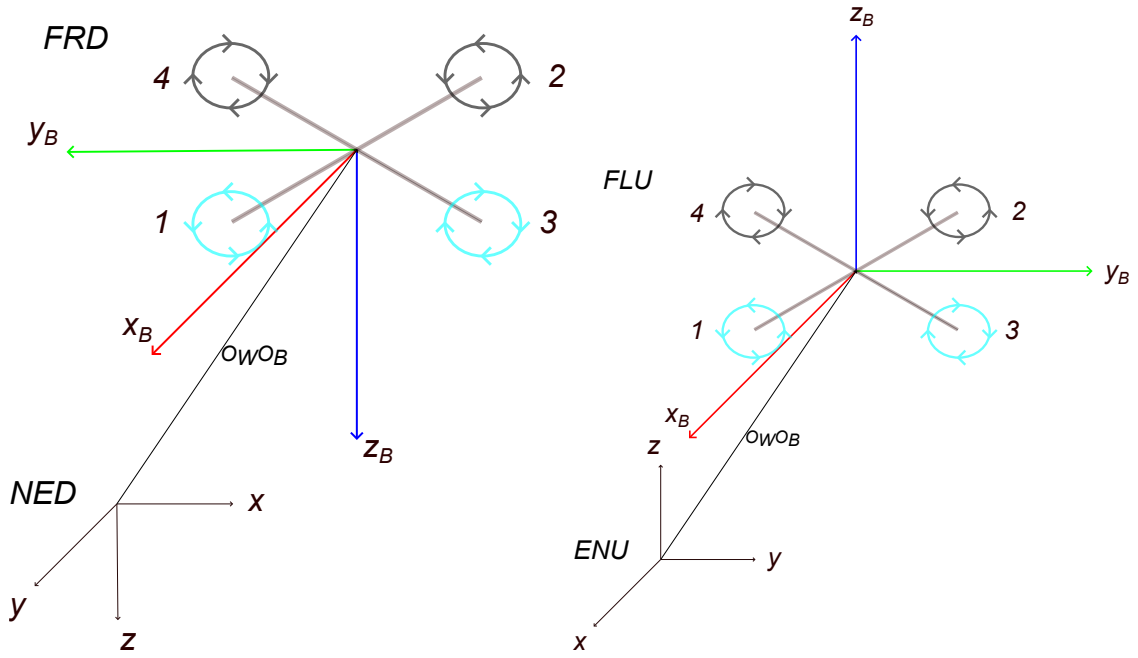


Figure 5.7: (a) PX4 reference frames, (b) MATLAB and ROS 2 reference frames. The light blue arrows define the front propellers of the vehicle. Adapted from [37].

The last block that is illustrated in this subsection is the one that converts all odometry data from the PX4 frames to the model ones. This conversion is a simple application of vector rotation and rotation combination using quaternions like in (2.15) and (2.14). The quaternions that encode the change between conventions can be derived from the intuitive XYZ Euler angles. The displacement to go from world NED to world ENU is given by $(\pi, 0, -\frac{\pi}{2})$ and to go from body FRD to body FLU is given by $(\pi, 0, 0)$.

6 Simulations

This chapter provides a description of the simulation environment then the setups and control parameters used for all tests are given. Finally, the results are reported and commented on.

6.1 Setup

PX4 software supports out-of-the-box both SITL and HITL simulations, for the first stage of this work the former is preferred to validate the control scheme without risks.

The natural choice for the simulation environment is Gazebo™ [38] since it is widely used in robotics applications because of its great customizability, open source code-base, good graphics and efficient physics engine. The communication scheme between Simulink, PX4 and the simulator is illustrated in Fig.6.1.

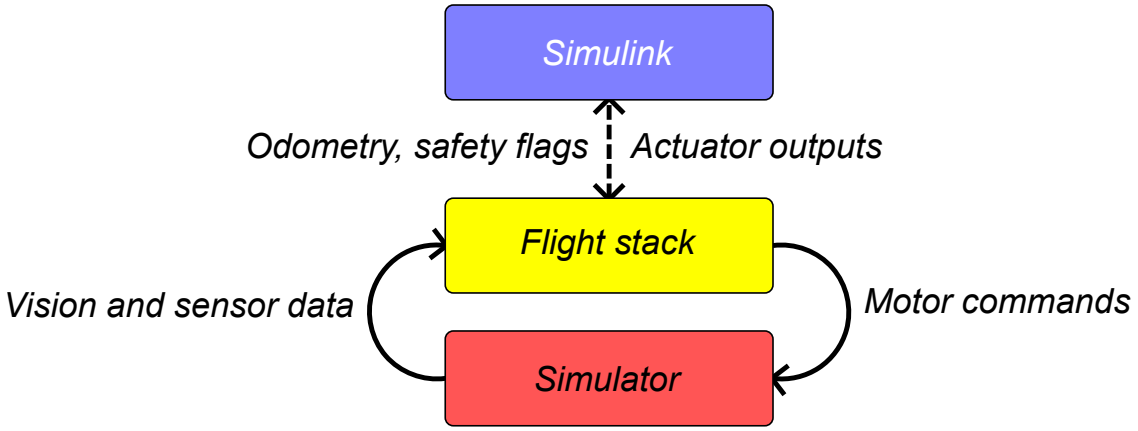


Figure 6.1: Scheme of the connection between MATLAB/Simulink, PX4 and Gazebo. Bold dotted lines represent communication through ROS 2, bold lines communication through the MAVLink API. Adapted from [39].

PX4 provides some ready-made Gazebo models for quad-copters, here the one named Iris (Fig.6.2) is used. Its main physical parameters are reported in Tab.6.1.

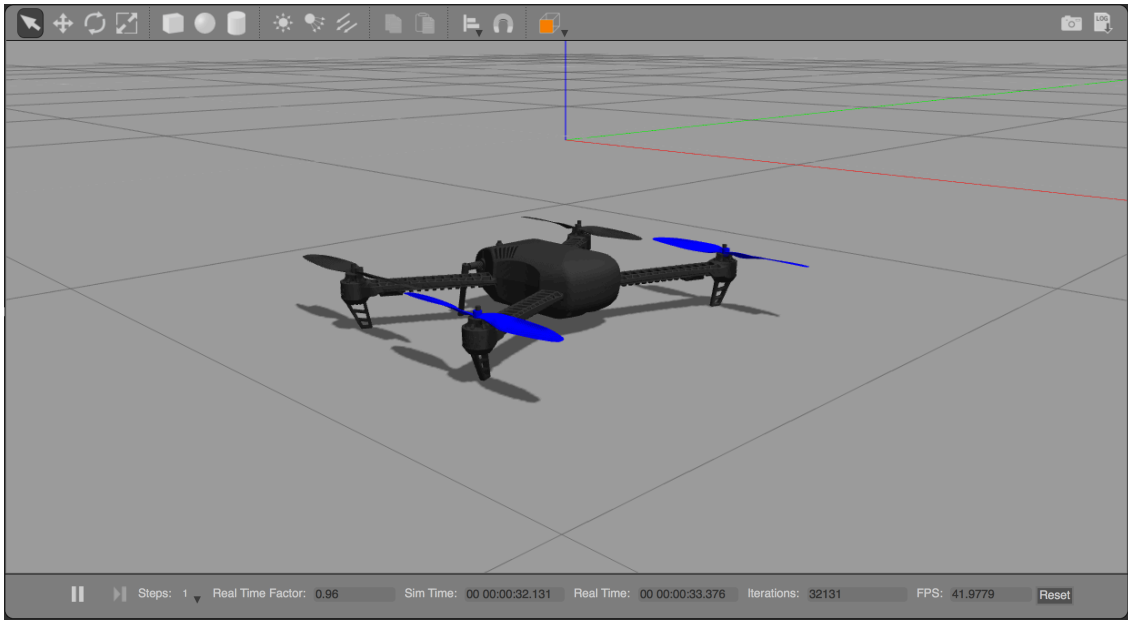


Figure 6.2: Iris model inside the Gazebo environment. Adapted from [40].

Iris parameters		
Name	Symbol [Unit]	Value
Mass	m [kg]	1.5
Arm length (front)	L_f [m]	0.255
Arm length (back)	L_b [m]	0.238
Propeller radius	r [m]	0.127
Moment of inertia	$\mathbf{J}$ [kg·m^2]$_{3\times3}$	diag[0.029, 0.029, 0.055]
Maximum thrust (per motor)	T_{max} [N]	5.25
Thrust coefficient	c_t [kg·m]	$4.28 \cdot 10^{-6}$
Moment coefficient	c_τ [m]	0.06

Table 6.1: Physical parameters of the Iris quad-rotor model.

The weight and parameters of the control system are kept constant across all simulations and are presented in Tab.6.2. Note that the main objective of this thesis is to validate the feasibility of the control scheme when paired with the PX4

system and not to tune both controllers for optimal performance.

Control scheme parameters		
Name	Symbol [Unit]	Value
Time step	T_s [s]	0.01
Step size	N	30
Position error weight	Q_p	[35, 30, 30]
Orientation error weight	Q_q	[1, 1, 15]
Speed error weight	Q_v	[5, 5, 5]
Angular speed error weight	Q_ω	[1.1, 1.1, 1.1]
Control output error weight	Q_u	[0.8, 0.8, 0.8, 0.8]
Roll constraint	$\epsilon_{x_{min}}, \epsilon_{x_{max}}$	-0.7071, 0.7071 ($\pm 45°$)
Pitch constraint	$\epsilon_{y_{min}}, \epsilon_{y_{max}}$	-0.7071, 0.7071 ($\pm 45°$)
Control output constraints	T_{min}, T_{max} [N]	0, 5.25
Adaptive gain matrix	A_s	$-$diag[20, 20, 10, 30, 30, 40]
Low-pass filter cut-off frequency	ω_{c0} [Hz]	0.628

Table 6.2: Parameters of NMPC and $\mathcal{L}_1$ control schemes.

6.1.1 Ground effect

Gazebo does not provide by default complex aerodynamics interactions so to make the simulation more realistic, without introducing a complex system that would be outside the scope of this work, a simple model for ground effect is considered and detailed below.

Ground effect is an aerodynamic phenomenon that happens when rotors spin parallel and at a close distance to the ground, the generated air pressure reduces the induced drag of the vehicle and therefore increases its lift-to-drag ratio, meaning that for the same motor output more thrust is generated by the propellers. This creates lots of problems in scenarios that require precise taking off and landing and has been studied for many years, particularly in the context of helicopters [41], and later adapted to quadrotors by introducing a correction factor to account for complex interactions between the air pressure generated by all four propellers working in tandem [42].
This model, reported below, provides an expression for the relation between upward thrust input $\mathbf{T}_{IN} = \begin{bmatrix} 0 & 0 & T_{IN} \end{bmatrix}^\top$ and real output $\mathbf{T}_{OUT} = \begin{bmatrix} 0 & 0 & T_{OUT} \end{bmatrix}^\top$, both expressed w.r.t. the world-frame

$$T_{OUT} = \frac{T_{IN}}{\left(1 - \rho(\frac{r}{4p_z})^2\right)} \tag{6.1}$$

Where $\rho \in \mathbb{R}^+$ is the correction factor, r is the radius of the propellers and p_z is the altitude of the quad-rotor. As can be seen from the formula and from Fig.6.3 as the altitude increases the ground effect disappears and $\mathbf{T}_{OUT} = \mathbf{T}_{IN}$.

The inclusion of the ground effect in this work is made to add another condition in which the system is subjected to unmodeled disturbances to demonstrate the effects of adaptive control, so a precise representation of this phenomenon is of secondary importance.

The correction factor used in [42] was found by fitting a curve over experimental data and since in this section none is available the same $\rho = 8.6$ as the paper is used. Moreover, ground effect is considered significant only in the altitude range $0.5 < \frac{p_z}{r} < 3$.

Finally, since in the model (6.3) only a hovering task is considered, a more general formulation is adopted, where the x and y components of $\mathbf{T}_{IN}$ and of $\mathbf{T}_{OUT}$ are not always zero. Therefore, the implemented model instead of the full commanded thrust $\mathbf{T}_{IN}$ uses its projection on the world-z axis. This is motivated

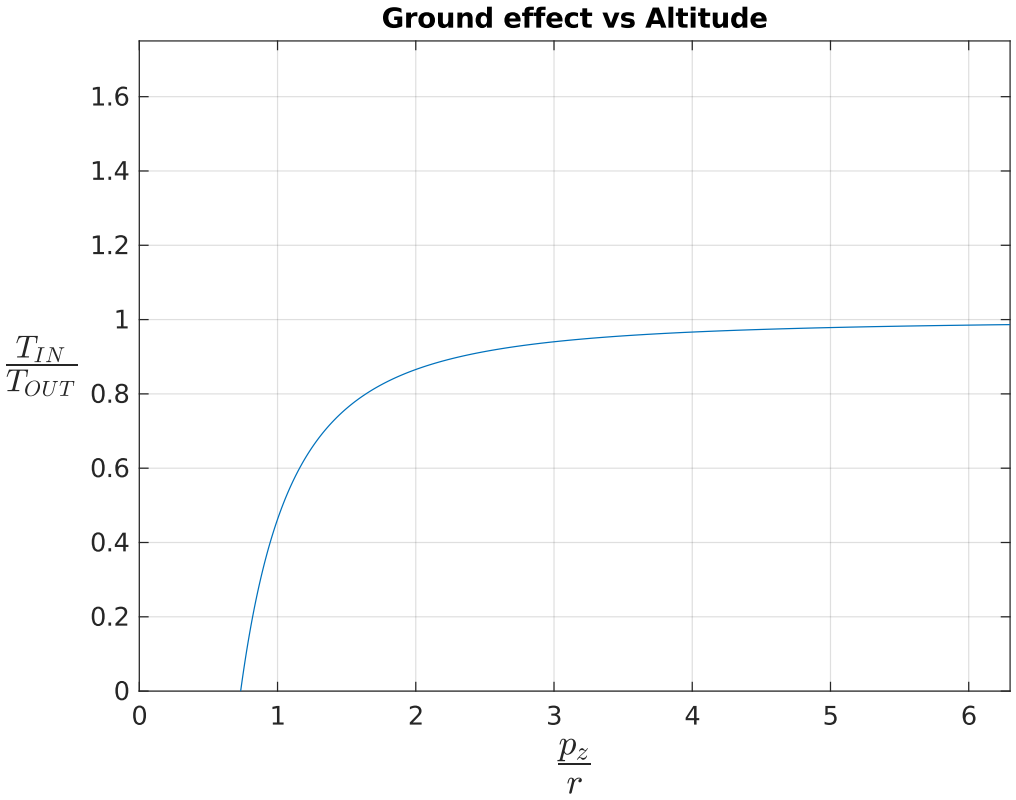


Figure 6.3: Ground effect vs altitude over propeller size. Adapted from [42].

by the fact that because ground effect is generated by a pocket of air pressure under the drone, it makes intuitive sense that it should get smaller if the drone is pushing air in another direction so that ideally, it is at its maximum and minimum respectively when the quad-copter is parallel and at 90° w.r.t. the ground.
The new model is

$$\mathbf{T}_{OUT} = \begin{bmatrix} \mathbf{T}_{IN} \cdot \hat{\mathbf{i}} \\ \mathbf{T}_{IN} \cdot \hat{\mathbf{j}} \\ \frac{1}{\left(1-\rho(\frac{r}{4p_z})^2\right)} \mathbf{T}_{IN} \cdot \hat{\mathbf{k}} \end{bmatrix} \tag{6.2}$$

Where $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, $\hat{\mathbf{k}}$ are the usual versors of the world-frame axes and $\cdot$ represent the standard scalar product between vectors.

To include this model in the calculations that gazebo makes to update motor thrust and torques it is sufficient to modify `PX4-SITL_gazebo-classic/src/gazebo_motor_model.cpp` to add a new positive force in the world-z axis applied at the drone's CoM.

6.2 Results

In this section, several scenarios are considered with combinations of both different trajectories and modeling errors. All tasks start after takeoff and last 20s and it is very important to note that since there is a switch between PX4 position control and this work's control scheme, the only part of the graphs that is relevant starts after the switch and ends 20s later when the control is switched back to PX4 for landing. In the error graphs the whole simulation time frame is reported since the error calculations by definition involve only the measurements during the Simulink control period.

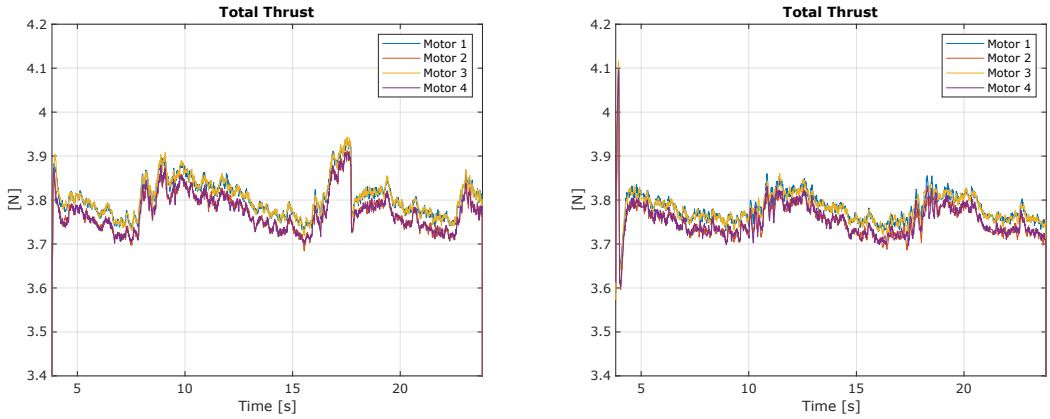
The considered tasks are illustrated in Tab.6.3. Every trajectory has a null rotation reference, meaning that the drone has to always be aligned with Gazebo's fixed reference frame. Every modeling error is applied when the control switches to Simulink.

Simulation scenarios						
Trajectory	Condition					
Hover	Ground effect					
	$\mathcal{L}_1$ off		$\mathcal{L}_1$ on			
Lemniscate	Nominal		Additional Payload		Damaged motor	
	$\mathcal{L}_1$ off	$\mathcal{L}_1$ on	$\mathcal{L}_1$ off	$\mathcal{L}_1$ on	$\mathcal{L}_1$ off	$\mathcal{L}_1$ on
Fast lemniscate	Additional Payload					
	$\mathcal{L}_1$ off		$\mathcal{L}_1$ on			

Table 6.3: Table with the considered simulation scenarios.

6.2.1 Hover with ground effect

The trajectory considered in this subsection is a simple hovering task at a constant altitude of 0.3m above the origin. At this altitude, ground effect is still relevant and disturbs the output of the NMPC controller, as illustrated in Fig.6.4.



(a) (b)

Figure 6.4: Hover task with ground effect: thrust output with (a) $\mathcal{L}_1$ off, (b) $\mathcal{L}_1$ on.

It can be seen from the Fig.6.4a that NMPC output oscillates because given a thrust command, it gets combined to ground effect creating peaks in the total force on the quadrotor, thus throwing off the predicted state and overshooting the altitude target. Then the controller tries to compensate by lowering the commanded thrust and thus going below the target. At a lower altitude, the ground effect gets stronger and therefore the cycle repeats. With the addition of $\mathcal{L}_1$ control Fig.6.4b this behavior is mitigated and the commanded thrust output is smoother.

Hover position tracking error [m]		
Nominal	$\mathcal{L}_1$ off	$\mathcal{L}_1$ on
RMSE	0.0553	0.0577

Table 6.4: Table with the Euclidean RMSE of hovering task.

Comparing the top-right graph in Fig.6.5 to the top-right graph in Fig.6.6 one can clearly see the effect of adaptive control on the speed error graph, where the z component still presents oscillations but with much-reduced amplitude. This behavior is reflected in the altitude error in the top-left graphs. Even though the total Euclidean tracking RMSE (Tab.6.4) is slightly higher, the values remain acceptable in a hovering task and the improved stability represents a valid point in favor of adaptive control.

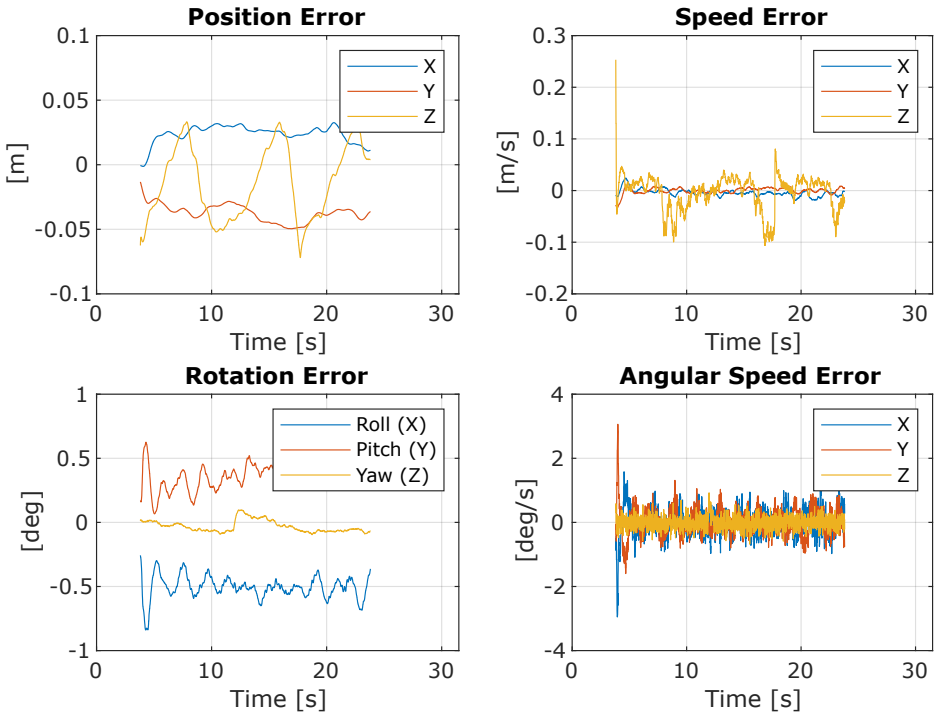


Figure 6.5: Errors w.r.t. references in hover task with ground effect and $\mathcal{L}_1$ off.

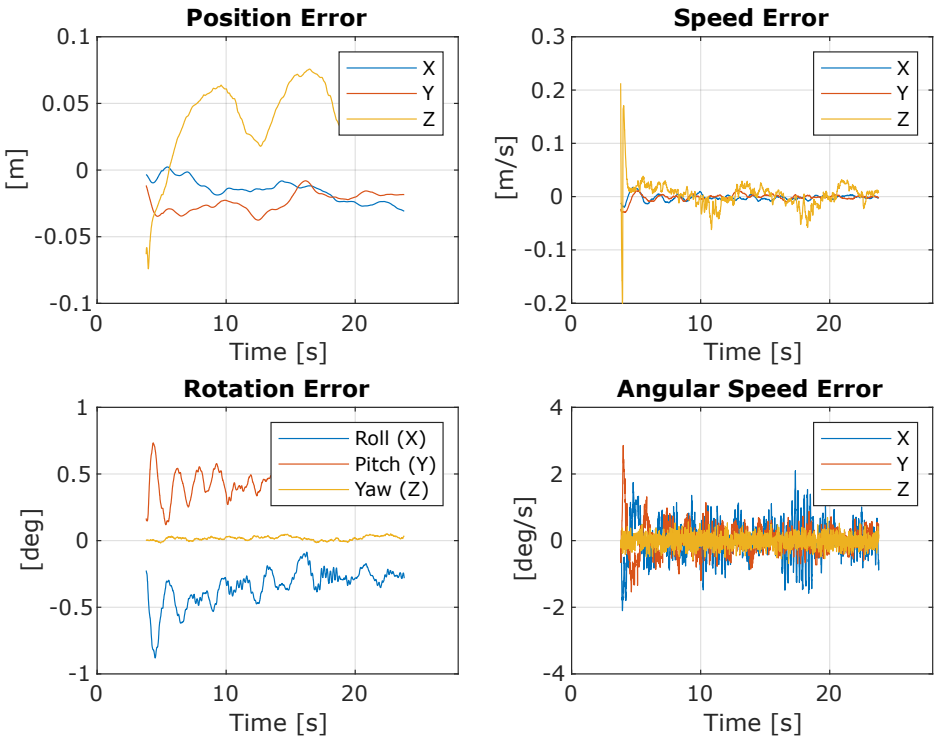


Figure 6.6: Errors w.r.t. references in hover task with ground effect and $\mathcal{L}_1$ on.

6.2.2 Lemniscate

In this subsection, the chosen trajectory is basically an eight-shaped curve called lemniscate. It was chosen because it allows the evaluation of both curvilinear and rectilinear movements, it is easy to derivate to get the reference speeds and accelerations and it is also often used as a benchmark in the literature. The one used in these simulations has a maximum linear velocity of 2m/s and has the following form

$$\mathbf{p}_{ref}(t) = \begin{bmatrix} 4sin(0.3535t) \\ 2sin(0.707t) \\ 5 \end{bmatrix}$$

Nominal case

Below is presented the simulation that represents the nominal case, i.e. both simulated and modeled systems coincide.

Lemniscate position tracking error [m]		
Nominal	$\mathcal{L}_1$ off	$\mathcal{L}_1$ on
RMSE	0.2379	0.2215

Table 6.5: Table with the Euclidean RMSE of nominal lemniscate task.

Reference tracking errors in the $\mathcal{L}_1$ off and on cases are reported respectively in Fig.6.7 and 6.8. Note also that some tracking errors on orientation and angular speed in the roll and pitch directions are inevitable since the drone cannot create accelerations in the x and y directions without inclining itself w.r.t. the ground. Because this simulation is carried out with nominal conditions, it is expected that the adaptive control should not modify significantly the NMPC output since there are no unmodeled disturbances, and therefore the error trajectories should remain the same. This assumption is verified by the similar RMSE and by looking at the above two figures: every graph between Fig.6.7 and Fig.6.8 present mostly the same evolution.
This can also be seen more intuitively by looking at Fig.6.9a and Fig.6.9b, the colored paths are almost indistinguishable.

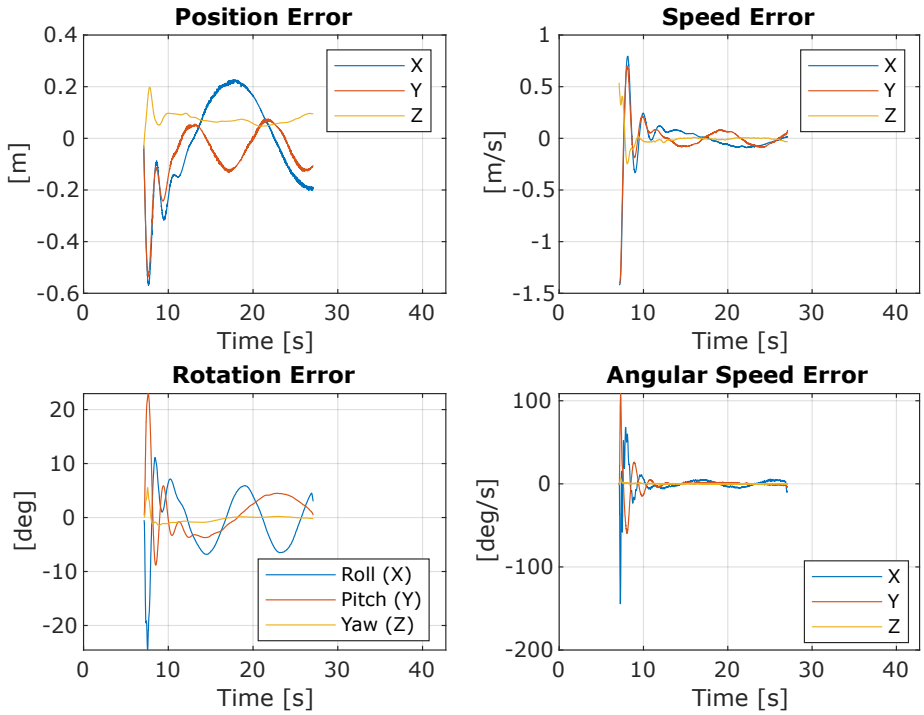


Figure 6.7: Errors w.r.t. references in lemniscate task, nominal conditions and $\mathcal{L}_1$ off.

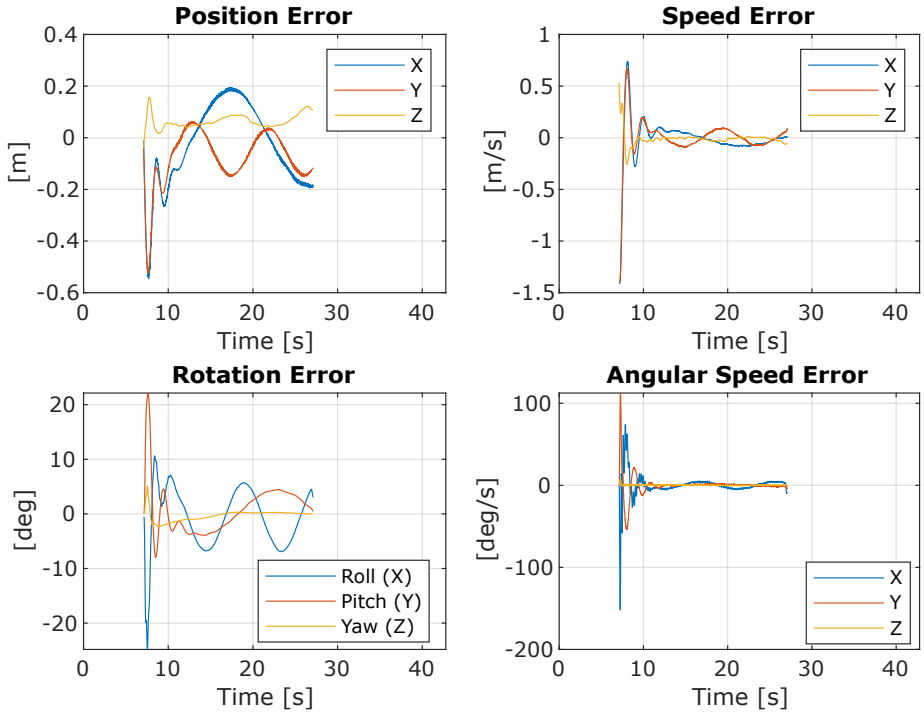
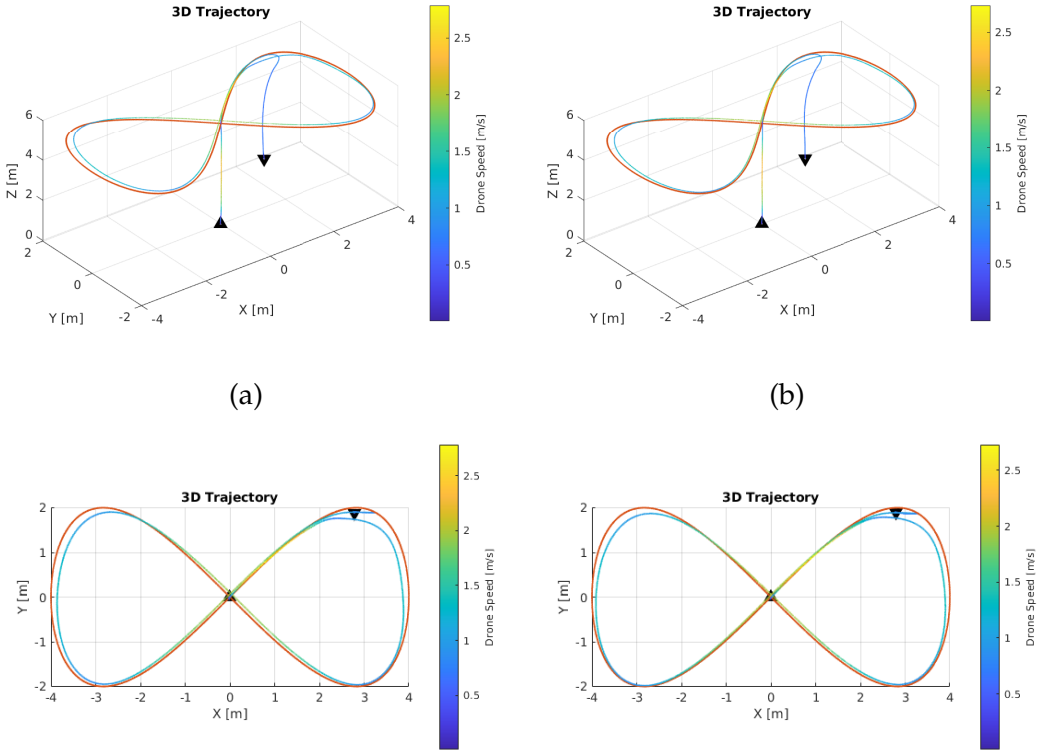


Figure 6.8: Errors w.r.t. references in lemniscate task, nominal conditions and $\mathcal{L}_1$ on.



(a) (b)

(c) (d)

Figure 6.9: 3d view of the trajectory in the upper figures, view from above in the lower two. Color represents linear speed magnitude. The reference is reported in red. (a), (c) $\mathcal{L}_1$ off; (b), (d) $\mathcal{L}_1$ on.

Additional payload case

This simulation represents the case in which an un-modeled payload is added to the drone right after the automated PX4 takeoff phase. The chosen additional mass is 0.5kg, which represents a 33% increase in the total mass of the quad-rotor. This is almost at the carrying limit of the vehicle since the motors can generate thrust to hover with a maximum additional mass of 0.67kg.

Lemniscate position tracking error [m]		
Additional payload	$\mathcal{L}_1$ off	$\mathcal{L}_1$ on
RMSE	0.6453	0.2403

Table 6.6: Table with the Euclidean RMSE of unmodeled additional payload lemniscate task.

Reference tracking errors in the $\mathcal{L}_1$ off and on cases are reported respectively in Fig.6.10 and Fig.6.11. In these graphs, it is evident that the big modeling error has a great impact on tracking performance, and the adaptive control contributes to the reduction of both tracking errors and vibrations due to the angular speed errors. Thanks to $\mathcal{L}_1$ control, tracking RMSE (reported in Tab.6.6) is comparable to the nominal case. It manages also to reduce the big oscillations around the body-x axis but the positions of the peaks remain the same, perhaps indicating some resonance in the drone's dynamics.
Looking at the 3d view of the trajectories (Fig.6.12) one can see that the main corrections given by adaptive contributions are, as expected, in the z direction to compensate for the higher weight.

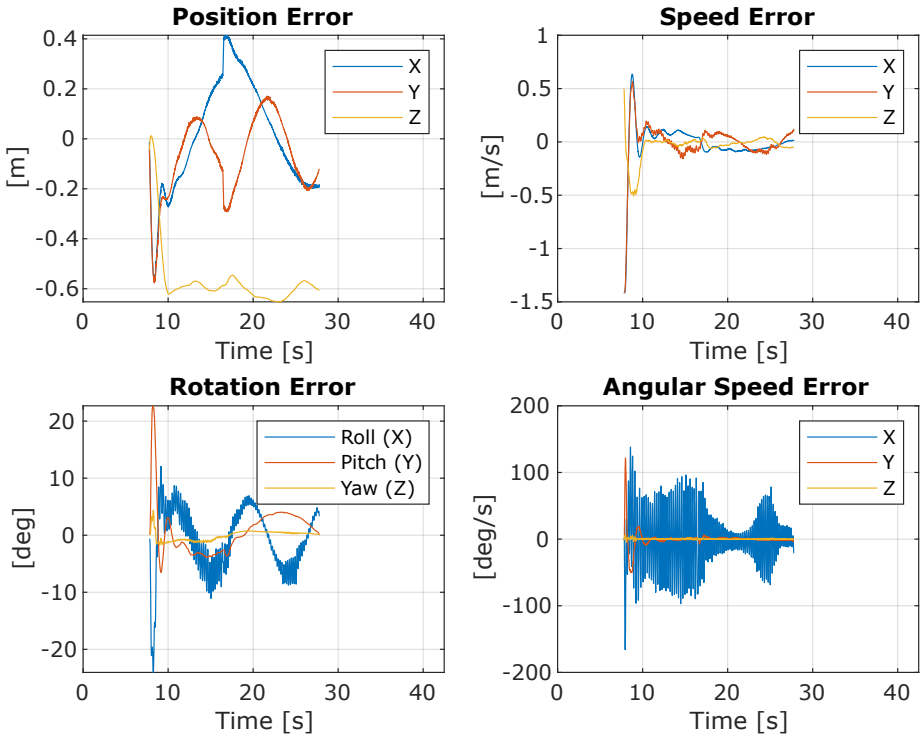


Figure 6.10: Errors w.r.t. references in lemniscate task, additional payload and $\mathcal{L}_1$ off.

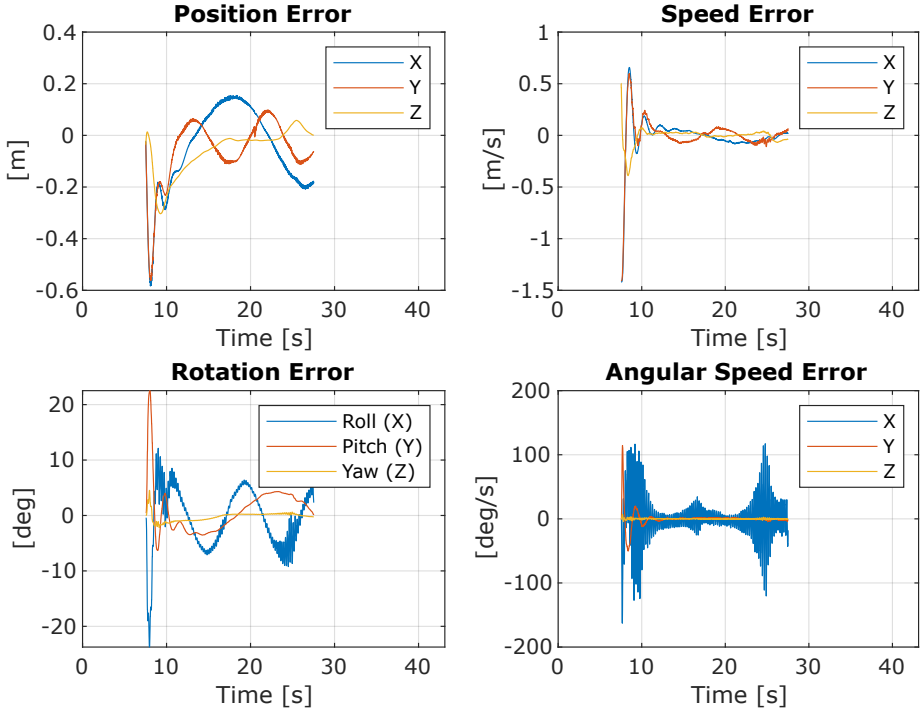
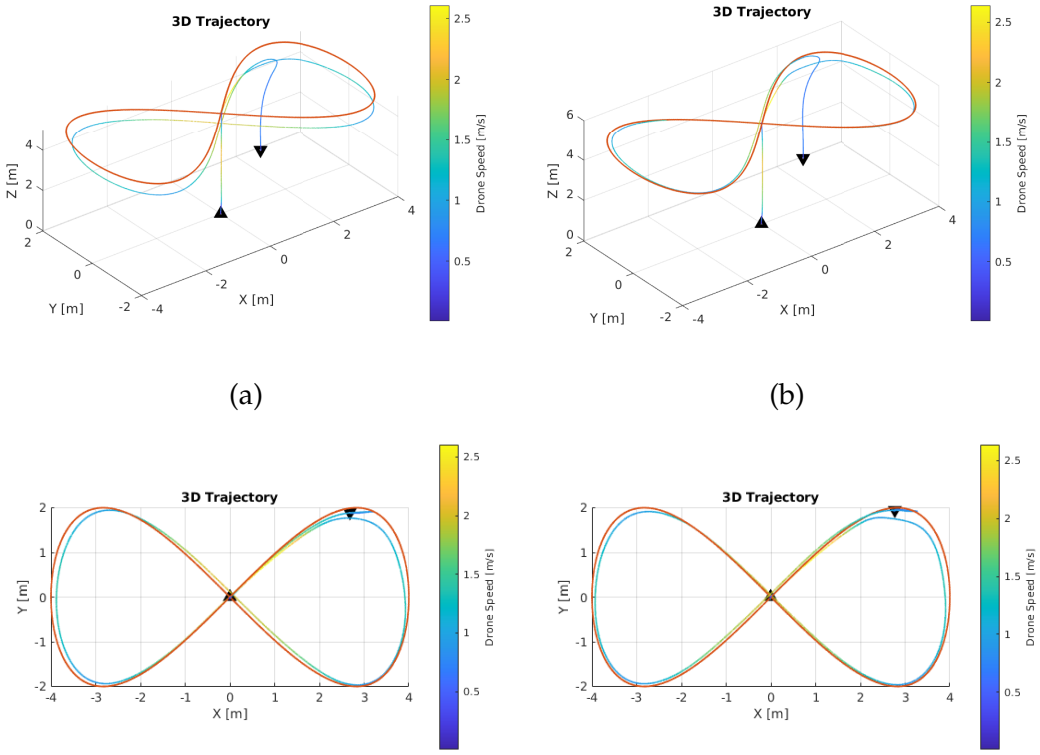


Figure 6.11: Errors w.r.t. references in lemniscate task, additional payload and $\mathcal{L}_1$ on.



(a) (b)

(c) (d)

Figure 6.12: 3d view of the trajectory in the upper figures, view from above in the lower two. Color represents linear speed magnitude. The reference is reported in red. (a), (c) $\mathcal{L}_1$ off; (b), (d) $\mathcal{L}_1$ on.

Partial motor failure

This simulation represents the case in which one of the motors is partially broken or, more realistically, partially blocked by dirt or other particulates that can get stuck in the rotor and the motor casing and thus reduce the torque output. Here the output thrust of one of the motors is limited to 70% of its original capabilities. This particular number was chosen since in the model of this drone actuator capabilities were already fairly limited and a greater reduction would result in an immediate crash. The motor limitation is applied only after the PX4 automated takeoff phase.

Lemniscate position tracking error [m]		
Partial rotor failure	$\mathcal{L}_1$ off	$\mathcal{L}_1$ on
RMSE	3.7034	0.5464

Table 6.7: Table with the Euclidean RMSE of partial motor failure lemniscate task.

Reference tracking errors in the $\mathcal{L}_1$ off and on cases are reported respectively in Fig.6.10 and Fig.6.11. From the graphs in the first figure, one can already see that they represent a critical situation, the drone is at less than $\frac{4}{5}$ of the target altitude and it is both vibrating and spinning violently while still trying to track the reference. When the simulation is repeated activating the adaptive control component the system is able to successfully recover from the failure while only retaining an offset in the orientation error. Nevertheless, it is able to track the trajectory with an RMSE (reported in Tab.6.7) that is more or less double w.r.t. the nominal case.
Looking at the 3d view of the trajectories (Fig.6.12) one can see clearly that without the $\mathcal{L}_1$ contribution the quad-copter almost crashes and spins around while trying to follow the prescribed path (6.15a-6.15c), notice also the great variation in speed during the spinning movement. Instead, with the additional contribution the drone is able to follow the lemniscate trajectory, albeit with some offset in the x direction.

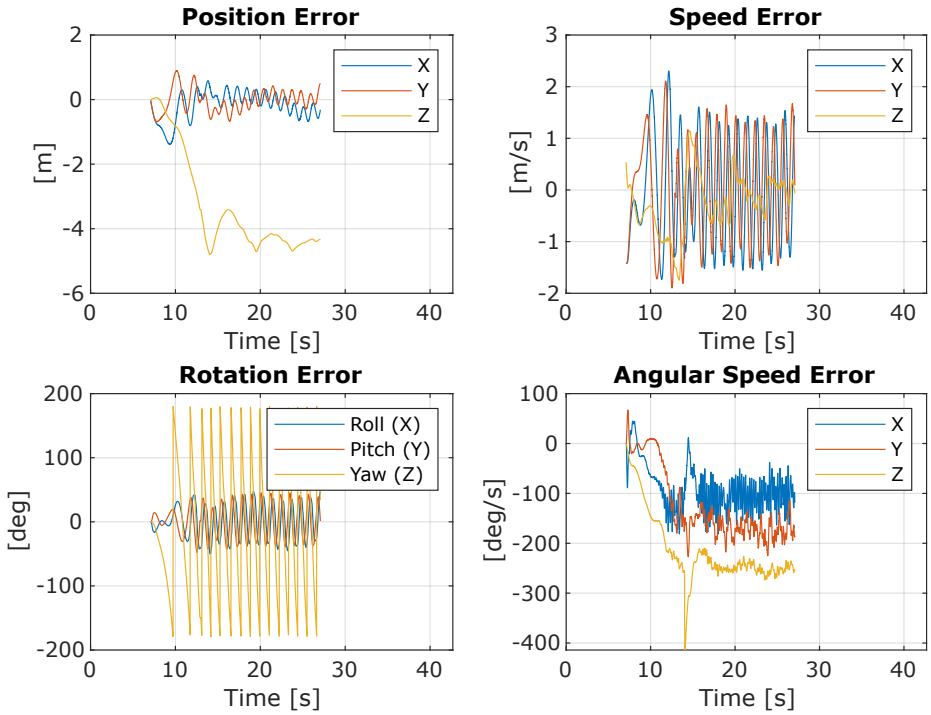


Figure 6.13: Errors w.r.t. references in lemniscate task, partial motor failure and $\mathcal{L}_1$ off.

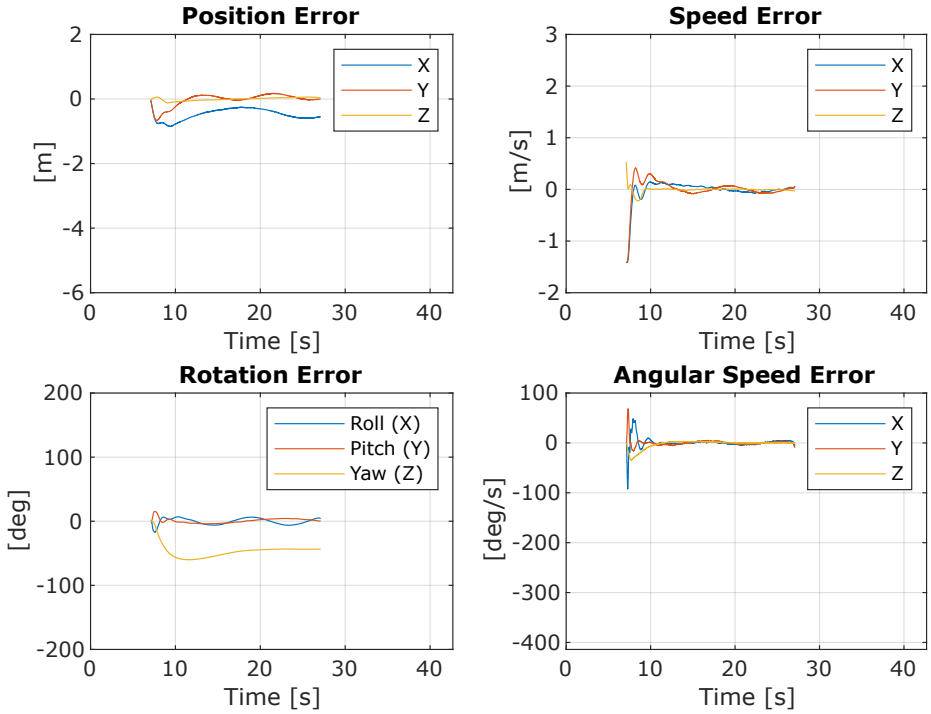
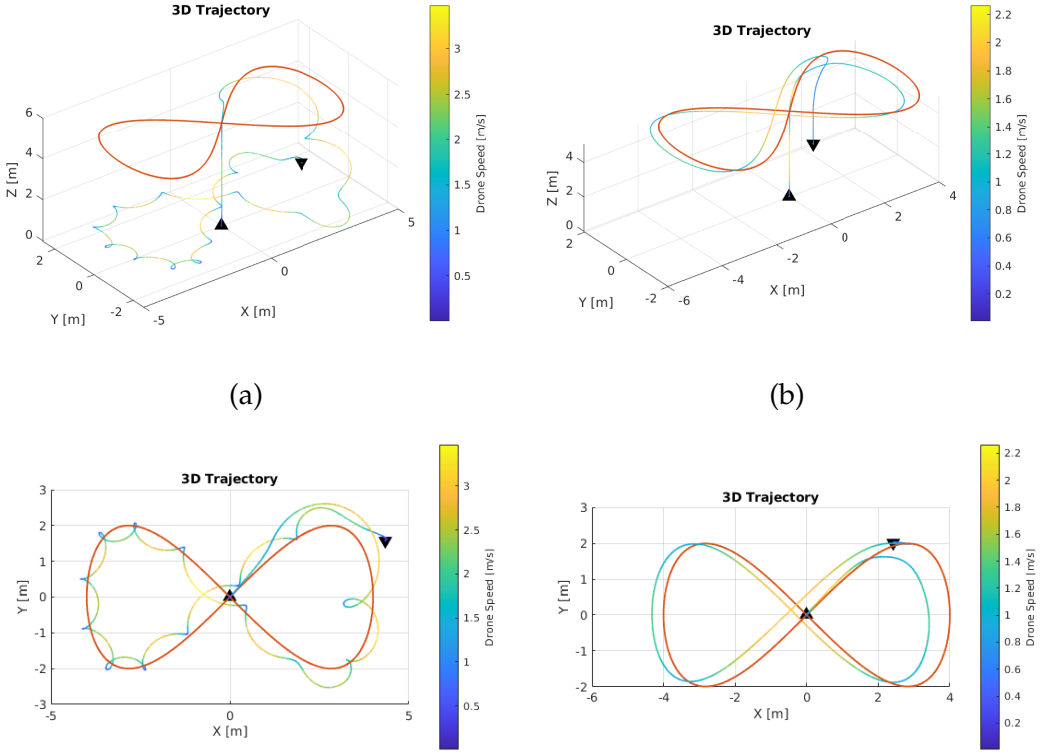


Figure 6.14: Errors w.r.t. references in lemniscate task, partial motor failure and $\mathcal{L}_1$ on.



(a) (b)

(c) (d)

Figure 6.15: 3d view of the trajectory in the upper figures, view from above in the lower two. Color represents linear speed magnitude. The reference is reported in red. (a), (c) $\mathcal{L}_1$ off; (b), (d) $\mathcal{L}_1$ on.

6.2.3 Fast lemniscate

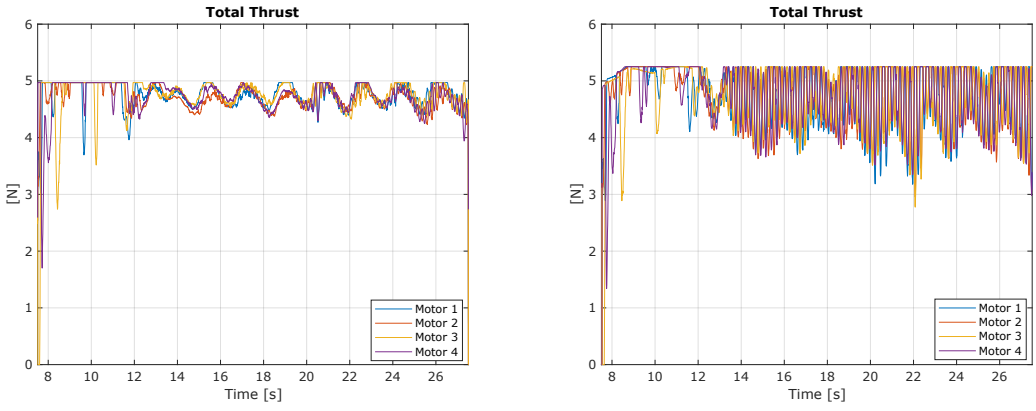
To test the limits of the system another simulation with increased payload is done with the difference being that this time the lemniscate trajectory has 2.5 times the maximum speed, for a total of 5m/s. This simulation is helpful in demonstrating the capability and usefulness of setting explicit hard control output limits in the model predictive control environment.

The new equation for the trajectory is

$$\mathbf{p}_{ref}(t) = \begin{bmatrix} 4sin(0.8838t) \\ 2sin(1.7676t) \\ 5 \end{bmatrix}$$

The additional mass parameters are the same as in the normal lemniscate case.

The motor limits can be clearly seen in Fig.6.16, in Fig.6.16a the NMPC is trying to optimize the output depending on the weights of the states and thrust reference errors while staying away from the limits as much as possible. In Fig.6.16b the $\mathcal{L}_1$ controller is ignoring the trade-off between output cost and state tracking errors, trying to only to minimize the latter and thus pushes the motor outputs right to their hard limits.



(a) (b)

Figure 6.16: Fast lemniscate with unmodeled mass task: thrust output with (a) $\mathcal{L}_1$ off, (b) $\mathcal{L}_1$ on.

Fast lemniscate position tracking error [m]		
Additional payload	$\mathcal{L}_1$ off	$\mathcal{L}_1$ on
RMSE	1.9004	1.2927

Table 6.8: Table with the Euclidean RMSE of unmodeled additional payload fast lemniscate task.

Reference tracking errors in the $\mathcal{L}_1$ off and on cases are reported respectively in Fig.6.17 and Fig.6.18 together with the RMSE in Tab.6.8. Once again looking at the graphs one can see that, although some fairly strong oscillations are introduced by $\mathcal{L}_1$ control, the adaptive scheme manages to lessen the error considerably.
Even more so than the last additional mass scenario, the drone is faced with the limits of its motors and being incapable of generating a high enough thrust to directly follow the trajectory it has to take a slight detour with both adaptive control off and on. After that initial dip in altitude, it is able to track the trajectory fairly well when adaptive control is activated, even though some significant vibrations around the body-x axis are introduced.
Looking at Fig.6.19, especially at Fig.6.19b and Fig.6.19d, it can be seen that the drone manages a good enough tracking after the initial undershoot in the z-component of the position. This would appear to be in contrast to the high RMSE reported in Tab.6.8, highlighting the problem with the metrics that keep track of the whole evolution of the error trajectory and not just of the steady-state error. In this case, the RMSE is swayed by the huge and rather lengthy initial dip in altitude.

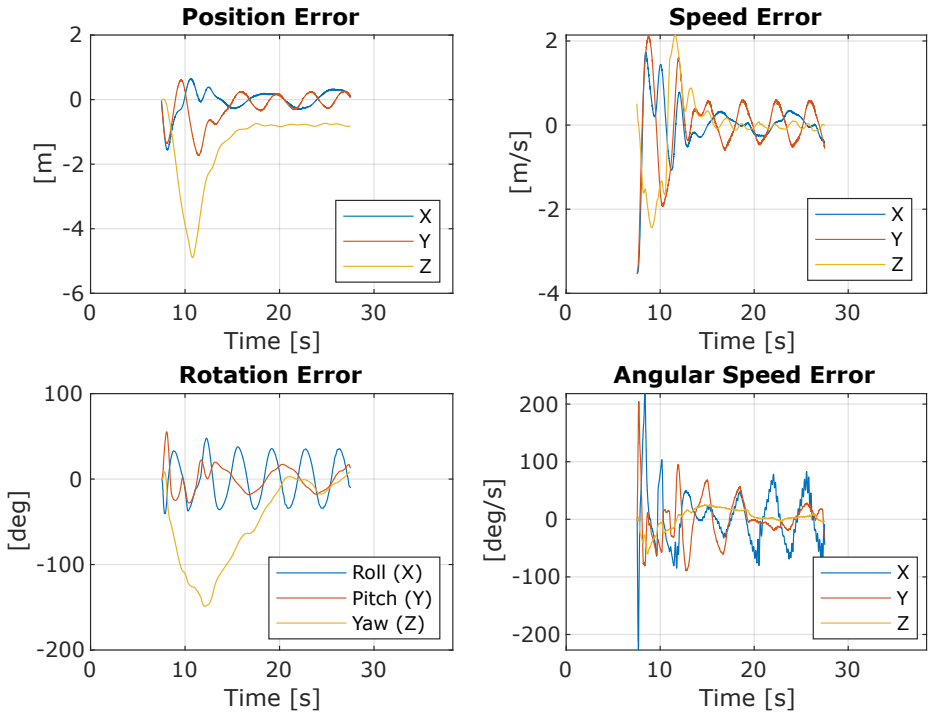


Figure 6.17: Errors w.r.t. references in lemniscate task, additional payload and $\mathcal{L}_1$ off.

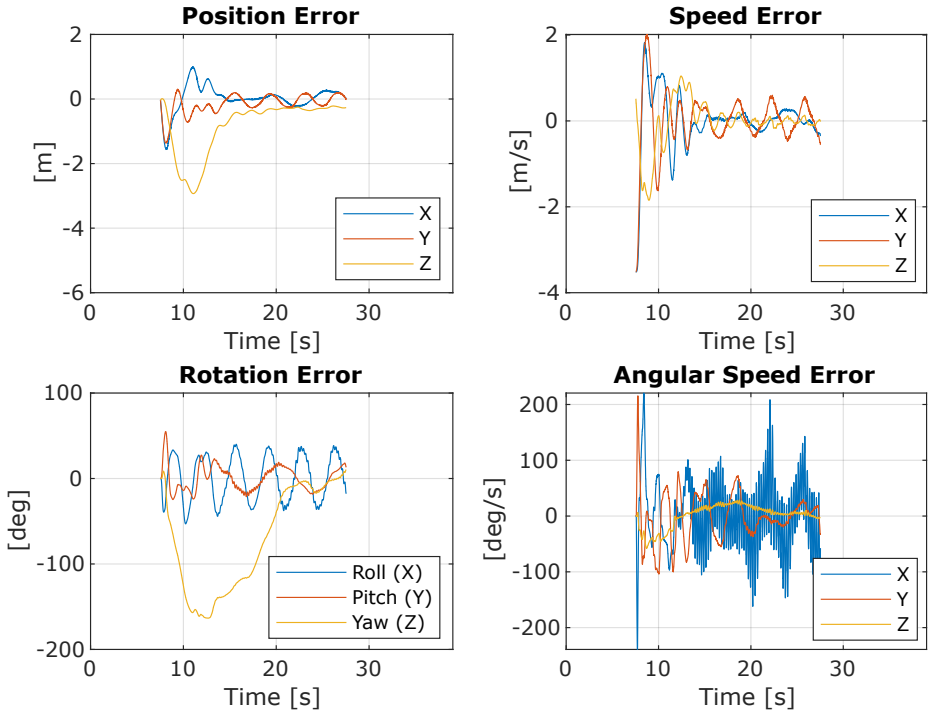


Figure 6.18: Errors w.r.t. references in lemniscate task, additional payload and $\mathcal{L}_1$ on.

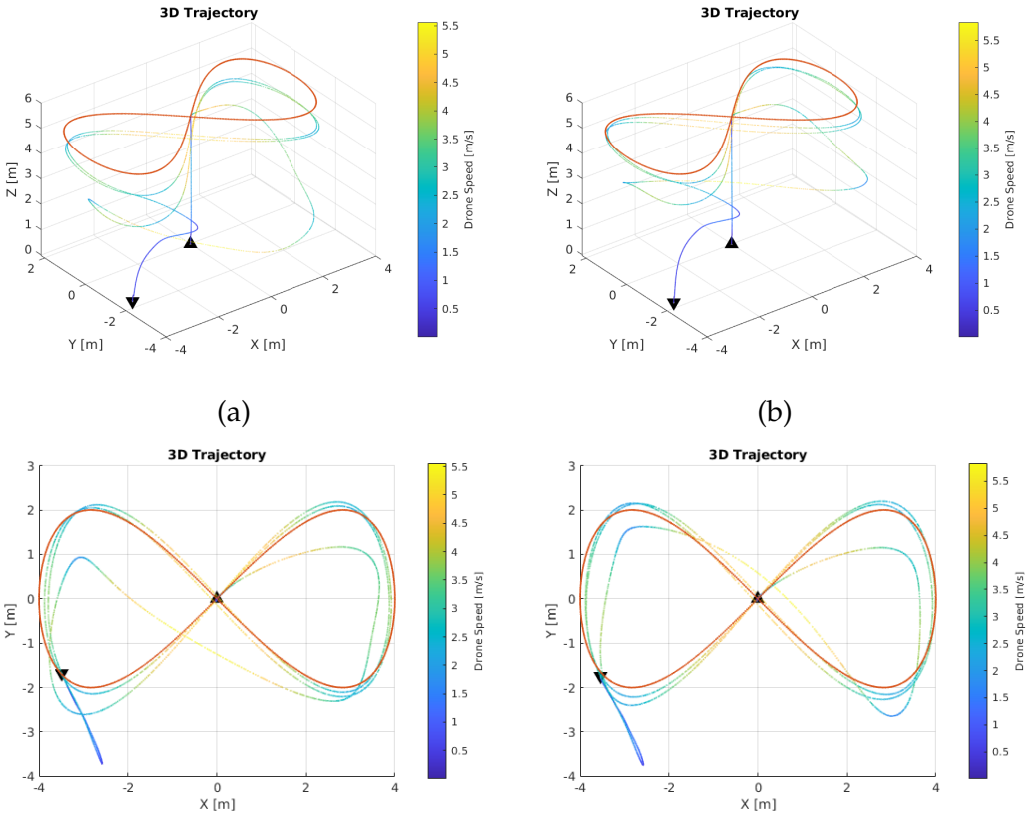


Figure 6.19: 3d view of the trajectory in the upper figures, view from above in the lower two. Color represents linear speed magnitude. The reference is reported in red. (a), (c) $\mathcal{L}_1$ off; (b), (d) $\mathcal{L}_1$ on.

7 Conclusions

The main contribution of this work is the creation of a Simulink scheme that allows the connection, through ROS2, of a PX4-enabled quad-copter and MATLAB. This expands enormously the capabilities of the drone, allowing much more functions and complex control algorithms to be effectively used, thanks to the greater computing power of a ground-based PC. This, coupled with the ease of use of the Simulink block scheme programming language allows the creation of a flexible yet powerful and easy-to-approach development environment. To demonstrate some of the possibilities given by this approach, in this work it is developed an implementation of nonlinear model predictive control then augmented with an independent adaptive contribution, since although very performant, the former is very susceptible to model inaccuracies that are inevitable in real-world applications. The control scheme is created using MATLAB/Simulink and then tested in real-time SITL simulation with various tasks and un-modeled conditions.

In Chp.2 starting from some considerations on the forces acting on a generic quad-copter its dynamical model is constructed using the Newton-Euler formalism. In Chp.3-4 there is a brief overview of the main ideas behind the concepts of adaptive and model predictive control, along with the specific modifications and adaptations needed in the UAVs framework. The complete infrastructure is then illustrated in Chp.5, starting from a recap of the main features and components of the PX4 software and of the other tools used. Then some comments on the NMPC, $\mathcal{L}_1$ and PX4 to ROS2 reference frame conversion Simulink blocks are made. Finally, the flowchart of the state machine that controls the communication between MATLAB and PX4 is given and explained. In Chp.6 all of the system physical and control parameters are presented; after a few comments on the Gazebo software the simulation environment is enriched with a ground effect model and the system is tested with several tasks. Every condition and trajectory is simulated both with and without the adaptive action to clearly understand its effects. The chosen conditions all represent important phenomena that are fairly common in real-world applications:

- Hovering is an important behavior for all UAVs and can give a lot of information regarding their stability, in this work the hovering task is carried out at a very low altitude to simulate the presence of ground effect, a disturbance that often causes problems in the takeoff, landing phases.

- Trajectory tracking is obviously the main objective of a vehicle that doesn't plan its own course and therefore is investigated using a lemniscate trajectory that enables the evaluation of both rectilinear and curvilinear movements. This trajectory is tested in two of the most important and frequent modeling error conditions: additional payloads and a partially broken rotor. A final test is carried out using a fast trajectory with additional mass to push the quad-rotor to its limits and showcase the explicit actuation constraint handling of the NMPC control.

The simulation results clearly demonstrate the effectiveness of the proposed control infrastructure. NMPC is able to predict future states and control the system such that the tracking results are satisfactory while in nominal conditions. When there are relevant modeling errors the adaptive component plays a major role in ensuring system stability and keeping the tracking performance at an acceptable level.

7.1 Future work

In this work the performance of the control scheme is assessed only in simulation, the connection infrastructure has been tested also in the lab with a HolyBro QAV250 quad-copter (Fig.7.1) and Vicon cameras as vision system but, due to time constraints, the physical parameters of its dynamical model could not be estimated and therefore the control scheme was highly unstable and the tests resulted in unusable data. In future works, this could be remedied thanks to simple and accurate enough estimation procedures, such as using a bifilar pendulum to estimate the drone's moments of inertia like in [43]. Finally, the combination of MATLAB/Simulink, ROS2 and PX4 software provides a very powerful and flexible environment that could open the doors to a much more vast set of applications. It could be used to control other types of vehicles and even more than one at the same time thanks to the networking capabilities of ROS2 and the baked-in safety and sensor management features of PX4, or it could be used in conjunction with the numerous autonomous guidance or machine learning toolboxes that MATLAB provides.

(a) (b)

Figure 7.1: HolyBro QAV250 quad-copter: on a table without the propellers (a), during flight in the flying arena in the SPARCS lab(b).

To facilitate future research and development the schemes and all code used in this thesis, along with a detailed written guide on installation and basic usage are made available on the SPARCS lab Gitlab page to future students.

A Time derivatives for rotations

A.1 Rotation matrices

Let $\mathbf{R(t)} \in \mathbb{SO}(3)$ be a rotation matrix. It holds that:

$$det(\mathbf{R}(t)) = 1 \tag{A.1a}$$

$$\mathbf{R}(t)\mathbf{R}^{\top}(t) = I_{3\times 3} \tag{A.1b}$$

Deriving (A.1b) with respect to time one gets

$$\dot{\mathbf{R}}(t)\mathbf{R}^{\top}(t) + \mathbf{R}(t)\dot{\mathbf{R}}^{\top}(t) = \mathbf{0}_{3\times 3} \tag{A.2}$$

Which indicates that $\mathbf{S}(t) \triangleq \dot{\mathbf{R}}(t)\mathbf{R}^{\top}(t)$ is a skew-symmetric matrix. Consequently it holds that

$$\dot{\mathbf{R}}(t) = \mathbf{S}(t)\mathbf{R}(t) \tag{A.3}$$

In this formulation the physical meaning of $\mathbf{S}(t)$ and how it contains information on angular velocity is not clear so a different derivation is preferred and given below.

Let $[\cdot]_{\times}$ be an operator called skew-symmetric operator that takes a vector $\boldsymbol{\omega} = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 \end{bmatrix}^{\top} \in \mathbb{R}^3$ and constructs a skew-symmetric matrix:

$$[\boldsymbol{\omega}]_{\times} \triangleq \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \in \mathbb{R}^{3\times 3} \tag{A.4}$$

This operator is useful because it allows to compute a cross product between vectors as a matrix product so that $\forall \boldsymbol{\omega}, \mathbf{x} \in \mathbb{R}^3$, $\boldsymbol{\omega} \times \mathbf{x} = [\boldsymbol{\omega}]_\times \mathbf{x}$. A particular property of rotation matrices, since they satisfy (A.1a) and (A.1b), is that

$$[\mathbf{R}(t)\boldsymbol{\omega}]_\times = \mathbf{R}(t)[\boldsymbol{\omega}]_\times \mathbf{R}^\top(t) \tag{A.5}$$

Now consider two reference frames $\mathcal{F}_W$ and $\mathcal{F}_B$, respectively fixed to the ground (world-frame) and fixed to a movable object (body-frame). In the following time dependence of all the vectors and matrices is omitted to simplify notation. Let $\mathbf{R}_{WB} = \mathbf{R}_{BW}^\top$ be the rotation matrix that encodes the rotation from frame $\mathcal{F}_B$ to $\mathcal{F}_W$ so that given any point, its description in body-frame $\mathbf{P}_B \in \mathbb{R}^3$ and the corresponding one in world-frame $\mathbf{P}_W \in \mathbb{R}^3$ can be found using

$$\mathbf{P}_W = \mathbf{R}_{WB}\mathbf{P}_B \tag{A.6a}$$

$$\mathbf{P}_B = \mathbf{R}_{BW}\mathbf{P}_W = \mathbf{R}_{WB}^\top \mathbf{P}_W \tag{A.6b}$$

Now let $\boldsymbol{\omega}_B \in \mathbb{R}^3$ be the angular velocity vector of $\mathcal{F}_B$ with respect to $\mathcal{F}_W$ expressed in $\mathcal{F}_B$, it represents the rate of rotation of the body frame ($\|\boldsymbol{\omega}_B\|$) and the direction along which it is rotating ($\frac{\boldsymbol{\omega}_B}{\|\boldsymbol{\omega}_B\|}$). Being a vector it can also be expressed in $\mathcal{F}_W$ using $\boldsymbol{\omega}_W = \mathbf{R}_{WB}\boldsymbol{\omega}_B$.

From a physical point of view the relationship between linear and angular speeds of an arbitrary point, described by the previously defined vectors $\mathbf{P}_W$ and $\mathbf{P}_B$ is given by

$$\dot{\mathbf{P}}_W = \boldsymbol{\omega}_W \times \mathbf{P}_W = [\boldsymbol{\omega}_W]_\times \mathbf{P}_W \tag{A.7}$$

While from a purely mathematical standpoint, since $\mathbf{P}_B$ is fixed in the body-frame and thus $\dot{\mathbf{P}}_B = 0$ it holds that

$$\dot{\mathbf{P}}_W = \dot{\mathbf{R}}_{WB}\mathbf{P}_B \tag{A.8}$$

By putting (A.7) in (A.8) it follows

$$\dot{\mathbf{R}}_{WB}\mathbf{P}_B = [\boldsymbol{\omega}_W]_\times \mathbf{P}_W = [\boldsymbol{\omega}_W]_\times \mathbf{R}_{WB}\mathbf{P}_B \tag{A.9}$$

It is clear that $\forall \mathbf{P}_B$

$$\dot{\mathbf{R}}_{WB} = [\boldsymbol{\omega}_W]_\times \mathbf{R}_{WB} \tag{A.10}$$

Finally in the context of robotics an IMU is often used to directly measure $\boldsymbol{\omega}_B$ and thus a more useful version of (A.10) can be derived by using property (A.5)

$$\begin{aligned}\dot{\mathbf{R}}_{WB} &= [\boldsymbol{\omega}_W]_\times \mathbf{R}_{WB} \\ &= [\mathbf{R}_{WB}\boldsymbol{\omega}_B]_\times \mathbf{R}_{WB} \\ &= \mathbf{R}_{WB}[\boldsymbol{\omega}_B]_\times \mathbf{R}_{WB}{}^\top \mathbf{R}_{WB} \\ &= \mathbf{R}_{WB}[\boldsymbol{\omega}_B]_\times\end{aligned} \tag{A.11}$$

For a more detailed explanation see [44].

A.2 Unit quaternions

Let $\mathbf{q}(t)$ be a unit quaternion composed by a real value η and a vector $\boldsymbol{\epsilon} \in \mathbb{R}^3$ so that $\mathbf{q}(t) = \begin{bmatrix} \eta(t) & \boldsymbol{\epsilon}^\top(t) \end{bmatrix}^\top$ with $\|\mathbf{q}(t)\| = 1$, used to encode a rotation between the same two reference frames $\mathcal{F}_W$ and $\mathcal{F}_B$ defined in the previous section. To simplify notation from now on time dependence will be omitted.

Let $\mathbf{q}_{WB}$ be the unit quaternion that represents the rotation from $\mathcal{F}_B$ to $\mathcal{F}_W$, $\widehat{\mathbf{P}_B} = \begin{bmatrix} 0 & \mathbf{P}_B^\top \end{bmatrix}^\top$ and $\widehat{\mathbf{P}_W} = \begin{bmatrix} 0 & \mathbf{P}_W^\top \end{bmatrix}^\top$ with $\mathbf{P}_B, \mathbf{P}_W \in \mathbb{R}^3$ the descriptions of the same arbitrary body-frame fixed point seen from $\mathcal{F}_B$ and $\mathcal{F}_W$ respectively. The two representations are related by

$$\begin{aligned} \widehat{\mathbf{P}_W} &= \mathbf{q}_{WB} \circ \widehat{\mathbf{P}_B} \circ \mathbf{q}_{BW} \\ &= \mathbf{q}_{WB} \circ \widehat{\mathbf{P}_B} \circ \overline{\mathbf{q}}_{WB} \end{aligned} \tag{B.12a}$$

$$\begin{aligned} \widehat{\mathbf{P}_B} &= \mathbf{q}_{BW} \circ \widehat{\mathbf{P}_W} \circ \mathbf{q}_{WB} \\ &= \overline{\mathbf{q}}_{WB} \circ \widehat{\mathbf{P}_W} \circ \mathbf{q}_{WB} \end{aligned} \tag{B.12b}$$

Where $\cdot \circ \cdot$ is the Hamilton product or quaternion composition operator and $\bar{\cdot}$ indicates a quaternion conjugate. Taking the derivative with respect to time of $\widehat{\mathbf{P}_W}$ in (B.12a) and substituting to $\widehat{\mathbf{P}_B}$ its expression in terms of $\widehat{\mathbf{P}_W}$ and knowing that $\widehat{\mathbf{P}_B}$ is fixed with respect to $\mathcal{F}_B$ one gets

$$\begin{aligned} \dot{\widehat{\mathbf{P}_W}} &= \dot{\mathbf{q}}_{WB} \circ \overline{\mathbf{q}}_{WB} \circ \widehat{\mathbf{P}_W} \circ \mathbf{q}_{WB} \circ \overline{\mathbf{q}}_{WB} + \mathbf{q}_{WB} \circ \dot{\overline{\mathbf{q}}}_{WB} \circ \widehat{\mathbf{P}_W} \circ \mathbf{q}_{WB} \circ \dot{\overline{\mathbf{q}}}_{WB} \\ &= \dot{\mathbf{q}}_{WB} \circ \overline{\mathbf{q}}_{WB} \circ \widehat{\mathbf{P}_W} + \widehat{\mathbf{P}_W} \circ \mathbf{q}_{WB} \circ \dot{\overline{\mathbf{q}}}_{WB} \end{aligned} \tag{B.13}$$

It holds that $\frac{d}{dt}(\mathbf{q}_{WB} \circ \overline{\mathbf{q}}_{WB}) = \dot{\mathbf{q}}_{WB} \circ \overline{\mathbf{q}}_{WB} + \mathbf{q}_{WB} \circ \dot{\overline{\mathbf{q}}}_{WB} = \mathbf{0}$. Developing the calculations for the first term of the sum and remembering that the norm is equal to one it follows that

$$\dot{\mathbf{q}}_{WB} \circ \overline{\mathbf{q}}_{WB} = \begin{bmatrix} 0 & -\dot{\eta}_{WB}\boldsymbol{\epsilon}_{WB} + \eta_{WB}\dot{\boldsymbol{\epsilon}}_{WB} - \dot{\boldsymbol{\epsilon}}_{WB} \times \boldsymbol{\epsilon}_{WB} \end{bmatrix}^\top = \begin{bmatrix} 0 & \boldsymbol{\nu}^\top \end{bmatrix}^\top \tag{B.14}$$

Therefore it also holds that $\mathbf{q}_{WB} \circ \dot{\overline{\mathbf{q}}}_{WB} = -\dot{\mathbf{q}}_{WB} \circ \overline{\mathbf{q}}_{WB} = \begin{bmatrix} 0 & -\boldsymbol{\nu}^\top \end{bmatrix}^\top$.

Using these facts in (B.13) yelds

$$\begin{aligned}\dot{\widehat{\mathbf{P}_W}} &= \begin{bmatrix}0\\ \boldsymbol{\nu}\end{bmatrix} \circ \widehat{\mathbf{P}_W} + \widehat{\mathbf{P}_W} \circ \begin{bmatrix}0\\ -\boldsymbol{\nu}\end{bmatrix} = \begin{bmatrix}0\\ \boldsymbol{\nu} \times \mathbf{P}_W\end{bmatrix} + \begin{bmatrix}0\\ \mathbf{P}_W \times (-\boldsymbol{\nu})\end{bmatrix} \\ &= \begin{bmatrix}0\\ \boldsymbol{\nu} \times \mathbf{P}_W\end{bmatrix} - \begin{bmatrix}0\\ -\boldsymbol{\nu} \times \mathbf{P}_W\end{bmatrix} \\ &= \begin{bmatrix}0\\ 2\boldsymbol{\nu} \times \mathbf{P}_W\end{bmatrix}\end{aligned} \tag{B.15}$$

Knowing from the previous section that from a physical standpoint (A.7) is true it immediately follows that

$$\widehat{\boldsymbol{\omega}_W} = \begin{bmatrix}0\\ \boldsymbol{\omega}_W\end{bmatrix} = \begin{bmatrix}0\\ 2\boldsymbol{\nu}\end{bmatrix} = 2\dot{\mathbf{q}}_{WB} \circ \overline{\mathbf{q}}_{WB} \tag{B.16}$$

Finally the derivative with respect to time of the quaternion that expresses a rotation from $\mathcal{F}_B$ to $\mathcal{F}_W$ is found with a right multiplication by $\mathbf{q}_{WB}$

$$\widehat{\boldsymbol{\omega}_W} \circ \mathbf{q}_{WB} = 2\dot{\mathbf{q}}_{WB} \Rightarrow \dot{\mathbf{q}}_{WB} = \frac{1}{2}\widehat{\boldsymbol{\omega}_W} \circ \mathbf{q}_{WB} \tag{B.17}$$

As said at the end of the previous section it is more useful in the context of robotics to define rotation derivatives in terms of body-frame angular velocity. To do that it is sufficient to rotate $\widehat{\boldsymbol{\omega}_W}$ following the same reasoning as in (B.12) and substitute the expression (B.16)

$$\begin{aligned}\widehat{\boldsymbol{\omega}_B} &= \overline{\mathbf{q}}_{WB} \circ \widehat{\boldsymbol{\omega}_W} \circ \mathbf{q}_{WB} \\ &= 2\overline{\mathbf{q}}_{WB} \circ \dot{\mathbf{q}}_{WB}\end{aligned} \tag{B.18}$$

And thus with a left multiplication by $\mathbf{q}_{WB}$ one gets the final expression for $\dot{\mathbf{q}}_{WB}$

$$\dot{\mathbf{q}}_{WB} = \frac{1}{2}\mathbf{q}_{WB} \circ \widehat{\boldsymbol{\omega}_B} = \frac{1}{2}\mathbf{q}_{WB} \circ \begin{bmatrix} 0 \\ \\ \boldsymbol{\omega}_B \end{bmatrix} \tag{B.19}$$

For a more detailed explanation see [45].

References

[1] D. Barcelos, A. Kolaei, and G. Bramesfeld, "Aerodynamic Interactions of Quadrotor Configurations," *Journal of Aircraft*, vol. 57, no. 6, pp. 1074–1090, Nov. 2020, Publisher: American Institute of Aeronautics and Astronautics, ISSN: 0021-8669. DOI: 10.2514/1.C035614. [Online]. Available: https://arc.aiaa.org/doi/10.2514/1.C035614.

[2] B. B. Carlos, T. Sartor, A. Zanelli, G. Frison, W. Burgard, M. Diehl, and G. Oriolo, *An Efficient Real-Time NMPC for Quadrotor Position Control under Communication Time-Delay*, arXiv:2010.11264 [cs, eess, math], Oct. 2020. DOI: 10.48550/arXiv.2010.11264. [Online]. Available: http://arxiv.org/abs/2010.11264.

[3] F. Nan, S. Sun, P. Foehn, and D. Scaramuzza, *Nonlinear MPC for Quadrotor Fault-Tolerant Control*, arXiv:2109.12886 [cs], Feb. 2022. DOI: 10.48550/arXiv.2109.12886. [Online]. Available: http://arxiv.org/abs/2109.12886.

[4] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: An engineering perspective," en, *The International Journal of Advanced Manufacturing Technology*, vol. 117, no. 5, pp. 1327–1349, Nov. 2021, ISSN: 1433-3015. DOI: 10.1007/s00170-021-07682-3. [Online]. Available: https://doi.org/10.1007/s00170-021-07682-3.

[5] J. Richalet, "Industrial applications of model based predictive control," en, *Automatica*, vol. 29, no. 5, pp. 1251–1274, Sep. 1993, ISSN: 0005-1098. DOI: 10.1016/0005-1098(93)90049-Y. [Online]. Available: https://www.sciencedirect.com/science/article/pii/000510989390049Y.

[6] R. Beard, N. Knoebel, C. Cao, N. Hovakimyan, and J. Matthews, "An L1 Adaptive Pitch Controller for Miniature Air Vehicles," vol. 8, Aug. 2006. DOI: 10.2514/6.2006-6777.

[7] M. Liu, F. Zhang, and S. Lang, "The Quadrotor Position Control Based on MPC with Adaptation," in *2021 40th Chinese Control Conference (CCC)*, ISSN: 1934-1768, Jul. 2021, pp. 2639–2644. DOI: 10.23919/CCC52363.2021.9549626.

[8] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, *L1-Adaptive MPPI Architecture for Robust and Agile Control of Multirotors*, arXiv:2004.00152 [cs, eess], Mar. 2020. DOI: 10.48550/arXiv.2004.00152. [Online]. Available: http://arxiv.org/abs/2004.00152.

[9] P. De Monte and B. Lohmann, "Position Trajectory Tracking of a Quadrotor based on L1 Adaptive Control," vol. 62, Apr. 2013, pp. 3346–3353. DOI: 10.1515/auto-2013-1035.

[10] X. Zhang, X. Li, K. Wang, and Y. Lu, "A Survey of Modelling and Identification of Quadrotor Robot," en, *Abstract and Applied Analysis*, vol. 2014, e320526, Oct. 2014, Publisher: Hindawi, ISSN: 1085-3375. DOI: 10.1155/2014/320526. [Online]. Available: https://www.hindawi.com/journals/aaa/2014/320526/.

[11] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, en. Nob Hill Publishing, 2017, Google-Books-ID: MrJctAEACAAJ, ISBN: 978-0-9759377-3-0.

[12] F. Alasali, S. Haben, H. Foudeh, and W. Holderbaum, "A Comparative Study of Optimal Energy Management Strategies for Energy Storage with Stochastic Loads," *Energies*, vol. 13, p. 2596, May 2020. DOI: 10.3390/en13102596.

[13] M. Morari and J. H. Lee, "Model predictive control: Past, present and future," en, *Computers & Chemical Engineering*, vol. 23, no. 4, pp. 667–682, May 1999, ISSN: 0098-1354. DOI: 10.1016/S0098-1354(98)00301-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098135498003019.

[14] M. A. Henson, "Nonlinear model predictive control: Current status and future directions," en, *Computers & Chemical Engineering*, vol. 23, no. 2, pp. 187–202, Dec. 1998, ISSN: 0098-1354. DOI: 10.1016/S0098-1354(98)00260-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098135498002609.

[15] A. Zanelli, G. Horn, G. Frison, and M. Diehl, "Nonlinear Model Predictive Control of a Human-Sized Quadrotor," Jun. 2018. DOI: 10.23919/ECC.2018.8550530.

[16] E. Hairer and G. Wanner, "RungeKutta Methods, Explicit, Implicit," en, in *Encyclopedia of Applied and Computational Mathematics*, B. Engquist, Ed., Berlin, Heidelberg: Springer, 2015, pp. 1282–1285, ISBN: 978-3-540-70529-1. DOI: 10.1007/978-3-540-70529-1_144. [Online]. Available: https://doi.org/10.1007/978-3-540-70529-1_144.

[17] D. Q. Huynh, "Metrics for 3D Rotations: Comparison and Analysis," en, *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155–164, Oct. 2009, ISSN: 1573-7683. DOI: 10.1007/s10851-009-0161-2. [Online]. Available: https://doi.org/10.1007/s10851-009-0161-2.

[18] Y. Wang, A. Ramirez-Jaime, F. Xu, and V. Puig, "Nonlinear Model Predictive Control with Constraint Satisfactions for a Quadcopter," vol. 783, Nov. 2016. DOI: 10.1088/1742-6596/783/1/012025.

[19] D. Hanover, P. Foehn, S. Sun, E. Kaufmann, and D. Scaramuzza, "Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 690–697, Apr. 2022, arXiv:2109.04210 [cs], ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2021.3131690. [Online]. Available: http://arxiv.org/abs/2109.04210.

[20] J. Van Amerongen, "Intelligent Control (part 1)-MRAS," *Lecture notes, University of Twente, The Netherlands*, 2004.

[21] C. Cao and N. Hovakimyan, "Design and Analysis of a Novel L1 Adaptive Controller, Part I: Control Signal and Asymptotic Stability," in *2006 American Control Conference*, ISSN: 2378-5861, Jun. 2006, pp. 3397–3402. DOI: 10.1109/ACC.2006.1657243.

[22] F. L. Lewis, "L1 Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation (Hovakimyan, N. and Cao, C.; 2010 [Bookshelf]," *IEEE Control Systems Magazine*, vol. 31, no. 5, pp. 112–114, Oct. 2011, Conference Name: IEEE Control Systems Magazine, ISSN: 1941-000X. DOI: 10.1109/MCS.2011.941837.

[23] E. Xargay, N. Hovakimyan, and C. Cao, "L1 adaptive controller for multi-input multi-output systems in the presence of nonlinear unmatched uncertainties," in *Proceedings of the 2010 American Control Conference*, ISSN: 2378-5861, Jun. 2010, pp. 874–879. DOI: 10.1109/ACC.2010.5530686.

[24] *Basic Concepts | PX4 User Guide*. [Online]. Available: https://docs.px4.io/main/en/getting_started/px4_basic_concepts.html.

[25] *Software Overview*, en-US. [Online]. Available: https://px4.io/software/software-overview/.

[26] *PX4/PX4-Autopilot at v1.14.0-beta1*. [Online]. Available: https://github.com/PX4/PX4-Autopilot/tree/v1.14.0-beta1.

[27] *PX4 Architectural Overview | PX4 User Guide*. [Online]. Available: https://docs.px4.io/main/en/concept/architecture.html.

[28] *ECL/EKF Overview & Tuning ů PX4 v1.9.0 User Guide*. [Online]. Available: https://docs.px4.io/v1.9.0/en/advanced_config/tuning_the_ecl_ekf.html.

[29] *Controller Diagrams | PX4 User Guide*. [Online]. Available: https://docs.px4.io/main/en/flight_stack/controller_diagrams.html.

[30] *uORB Messaging | PX4 User Guide*. [Online]. Available: https://docs.px4.io/main/en/middleware/uorb.html.

[31] *MAVLink Messaging | PX4 User Guide*. [Online]. Available: https://docs.px4.io/main/en/middleware/mavlink.html.

[32] *Flight Review*. [Online]. Available: https://review.px4.io/upload.

[33] *Overview ů QGroundControl User Guide*. [Online]. Available: https://docs.qgroundcontrol.com/master/en/index.html.

[34] *XRCE-DDS (PX4-FastDDS Bridge) | PX4 User Guide*. [Online]. Available: https://docs.px4.io/main/en/middleware/xrce_dds.html.

[35] Y. Chen, M. Bruschetta, E. Picotti, and A. Beghi, "MATMPC - A MATLAB Based Toolbox for Real-time Nonlinear Model Predictive Control," in *2019 18th European Control Conference (ECC)*, Jun. 2019, pp. 3365–3370. DOI: 10.23919/ECC.2019.8795788.

[36] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," en, *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, Mar. 2019, ISSN: 1867-2957. DOI: 10.1007/s12532-018-0139-4. [Online]. Available: https://doi.org/10.1007/s12532-018-0139-4.

[37] *Using Vision or Motion Capture Systems for Position Estimation | PX4 User Guide*. [Online]. Available: https://docs.px4.io/main/en/ros/external_position_estimation.html#reference-frames-and-ros.

[38] *Gazebo*. [Online]. Available: https://classic.gazebosim.org/.

[39] *Simulation | PX4 User Guide*. [Online]. Available: https://docs.px4.io/main/en/simulation/.

[40] *Gazebo Simulation ů PX4 Developer Guide*. [Online]. Available: https://dev.px4.io/v1.10_noredirect/en/simulation/gazebo.html.

[41] I. C. Cheeseman and W. E. Bennett, "The effect of the ground on a helicopter rotor in forward flight," en, 1955, Accepted: 2014-10-21T15:54:14Z. [Online]. Available: https://reports.aerade.cranfield.ac.uk/handle/1826.2/3590.

[42] L. Danjun, Z. Yan, S. Zongying, and L. Geng, "Autonomous landing of quadrotor based on ground effect modelling," in *2015 34th Chinese Control Conference (CCC)*, ISSN: 1934-1768, Jul. 2015, pp. 5647–5652. DOI: 10.1109/ChiCC.2015.7260521.

[43] J. Habeck and P. Seiler, "Moment of Inertia Estimation Using a Bifilar Pendulum," en, Apr. 2016, Accepted: 2016-10-03T13:15:46Z. [Online]. Available: http://conservancy.umn.edu/handle/11299/182514.

[44] S. Zhao, *Time Derivative of Rotation Matrices: A Tutorial*, arXiv:1609.06088 [cs], Sep. 2016. DOI: 10.48550/arXiv.1609.06088. [Online]. Available: http://arxiv.org/abs/1609.06088.

[45] B. Graf, *Quaternions and dynamics*, arXiv:0811.2889 [math-ph], Nov. 2008. DOI: 10.48550/arXiv.0811.2889. [Online]. Available: http://arxiv.org/abs/0811.2889.